



Calhoun: The NPS Institutional Archive
DSpace Repository

Theses and Dissertations

1. Thesis and Dissertation Collection, all items

1974-06

Machine independence: the problem of portability

Coulter, Edward Carl; Parker, Daniel James

Monterey, California. Naval Postgraduate School

<http://hdl.handle.net/10945/17111>

This publication is a work of the U.S. Government as defined in Title 17, United States Code, Section 101. Copyright protection is not available for this work in the United States.

Downloaded from NPS Archive: Calhoun



Calhoun is the Naval Postgraduate School's public access digital repository for research materials and institutional publications created by the NPS community. Calhoun is named for Professor of Mathematics Guy K. Calhoun, NPS's first appointed -- and published -- scholarly author.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

<http://www.nps.edu/library>

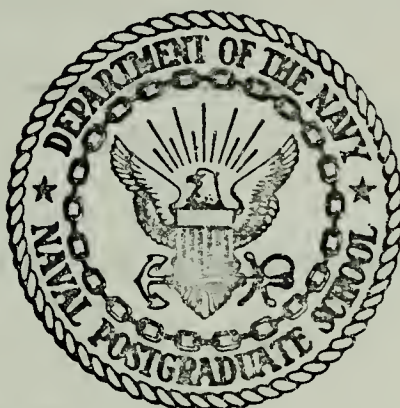
MACHINE INDEPENDENCE: THE PROBLEM OF
PORTABILITY

Edward Carl Coulter

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA 93940

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

MACHINE INDEPENDENCE
THE PROBLEM OF PORTABILITY

by

Edward Carl Coulter
and
Daniel James Parker

June 1974

Thesis Advisor:

G. A. Kildall

Approved for public release; distribution unlimited.

T 10, 045

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Machine Independence The Problem of Portability		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; June 1974
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Edward Carl Coulter and Daniel James Parker		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE June 1974
		13. NUMBER OF PAGES 96
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Naval Postgraduate School Monterey, California 93940		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Portability Machine Independence		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) Computer program portability is a measure of the effort which must be expended when transferring a program from one environment to another. Historically, there have been several approaches to portability, but most have met with little success. The problems of portability are presented, along with techniques available to computer programmers to solve or alleviate those problems. Concurrent with this study, a portable preprocessor/editor was designed. The preprocessor/editor was intended to		

20.

give FORTRAN users a machine-independent means of character output and provide a simple editor for text manipulation.

Machine Independence
The Problem of Portability

by

Edward Carl Coulter
Lieutenant, United States Navy
B.S., University of Arizona, 1967

and

Daniel James Parker
Lieutenant, United States Navy
B.A., San Diego State College, 1968

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1974

ABSTRACT

Computer program portability is a measure of the effort which must be expended when transferring a program from one environment to another. Historically, there have been several approaches to portability, but most have met with little success. The problems of portability are presented, along with techniques available to computer programmers to solve or alleviate those problems. Concurrent with this study, a portable preprocessor/editor was designed. The preprocessor/editor was intended to give FORTRAN users a machine-independent means of character output and provide a simple editor for text manipulation.

TABLE OF CONTENTS

I.	INTRODUCTION	8
	A. RESEARCH GOALS	8
	B. TERMS AND DEFINITIONS	8
	1. Bootstrapping	9
	2. Macro	9
	3. Translator	9
II.	MACH PROCESSORS	11
	A. THE MOBILE PROGRAMMING SYSTEM	11
	1. SIMCMP	12
	2. STAGE2	12
	a. Storage	13
	b. Arithmetic Expression Evaluation	13
	c. Conditional and Loop Operations	13
	B. LIMP	14
	1. Parameters	15
	2. Trees	15
III.	HIGH-LEVEL LANGUAGE PROGRAMMING	17
	A. STANDARDIZATION	17
	B. STRUCTURED PROGRAMMING	19
IV.	PREDICTORS	21
	A. INDEPENDENT ASPECTS	21
	B. DEPENDENT ASPECTS	22
	C. METHODS OF TRANSFER	22
	D. OPERATING PROCEDURES	23
	E. \$\$ STATEMENTS	25
	F. \$ CONTROL STATEMENTS	26
	1. \$OUTPUT	26
	a. Strings	26
	b. Integers	27
	2. \$COPY	27
	3. \$APPEND VERSE	28
V.	SUMMARY	29

APPENDIX A	FORMAL DEFINITION OF PREDITOR	31
APPENDIX E	PREDITOR CONTROL TOGGLES	32
APPENDIX C	ERROR MESSAGES	33
APPENDIX D	FILE DEFINITIONS	34
SAMPLE COMPUTER OUTPUT	35
PREDITOR LISTING	48
LIST OF REFERENCES	94
BIBLIOGRAPHY	95

LIST OF FIGURES

Figure		Page
1.	Typical PREDITOR Source File	24
2.	Typical PREDITOR Patch File	24

I. INTRODUCTION

Of major concern to the computer industry in recent years has been what W. M. Waite [Ref. 1] describes as the "Software Crises." That is, the problem of reprogramming when the need arises to transport software from one machine to another must be given major consideration as a factor in computer program portability. The ease with which the transfer is accomplished is measured by two factors, portability and adaptability. Portability can be defined as a measure of the ease with which a program can be transferred from one environment to another, while adaptability is a measure of ease with which a program can be altered to fit differing user applications [Poole and Waite, 1973]. Although the differences between the two are not always well defined in practice, the major distinction lies in the above definitions. Where portability deals with environmental changes, such as word length, adaptability is concerned with changes in algorithm structure.

A. RESEARCH GOALS

This research was an initial study of program portability. It was intended to present a broad picture of the problems of software transferability and provide some insight to how those problems could be solved. The programming of a preprocessor/editor was undertaken to provide some insight to the problems of portability while providing a facility to allow machine-independent FORTRAN based software.

B. TERMS AND DEFINITIONS

The following terms and their definitions are provided to give the reader a better understanding of the terminology used.

1. Bootstrapping

As used here, bootstrapping is a process commonly used in compiler implementation. The process is used when a compiler for a source language L (or a subset of L) is implemented on a particular machine and the compiler is to be written in the language L. Gries [Ref. 2] offers the following explanation in the form of an exercise:

Write a compiler in assembly language for a small subset L[0] of L. This subset should be small, so that it is easy to implement, but large enough to be used in the next step. The next step is to rewrite the compiler for L[0] in itself and check it out. Now, we try to "bootstrap" our way up to L in a series of steps as follows. At each step i, $i = 1, \dots, n$, extend the compiler for L[i-1] to a compiler for a language L[i], by implementing other features of the desired language L.

2. Macro

A macro gives the programmer the ability to refer to a specific code sequence by mentioning a single predefined name. Waite [Ref. 3] defines macros as having a form similar to the following:

MACRO NAME (P1,P2,...,Pn)

Code Body

END

The words "MACRO" and "END" serve to delimit the definition, "NAME" is the macro name, and "P1" through "Pn" are formal parameters. The code body is a series of lines which may contain instances of the formal parameters.

Unlike subroutine calls which occur at run time, macro expansion is the result of a preprocessing function. That is, when a macro call is recognized, actual parameters are substituted for formal parameters and the desired code is inserted at the point of the macro call.

3. Translator

McKeeman [Ref. 4] defines a translator as a "function whose domain is a source language and whose range is contained in a target language." Simply stated, it is a mapping of one language into another.

There are many factors which contribute to the problems of program portability. These factors may range from the manufacturer's unwillingness to produce compatible machines to the software community which, as stated by Fleiss, "has not provided the necessary guidelines or standards for program design, coding and documentation" [Ref. 5]. Because of these factors, one finds it necessary to develop portable software. Two approaches to solving the problems of portability are discussed: MACRO processors and high level language programming.

II. MACRO PROCESSORS

Various approaches to portability through the use of macro processors have been attempted in the past. As early as 1961 an attempt to enhance portability was undertaken in a project known as SLANG. This project was somewhat similar to another early approach, UNCOL. Both systems used translators and were based upon the premise that a translator can be written in a single intermediate language. Poole and Waite [Ref. 6] attribute the failure of these systems to their "simplistic" modeling. The variety of data bases and operators in existing languages necessitates a more comprehensive approach to abstract machine modeling. As Halpern [Ref. 7] points out, "with new computer applications being found daily, with the computer-using population growing at a still accelerating pace, and with that population taking on an increasingly lay character, the notion of a universal language is indefensible today." Since the computer world has somewhat settled on a multiplicity of languages, the above statement seems to hold. Processors on the other hand, are generally tailored to fit one language necessitating a multitude of compilers. The need is for a software processor which can handle the variety of languages which exist today and is somewhat extendable to meet the needs of future languages. To this end, the macro processor has been developed.

A. THE MOBILE PROGRAMMING SYSTEM

The Mobile Programming System is a translator system consisting of two levels, SIMCMP and STAGE2. The system is a "powerful macro processor designed specifically as a tool for constructing machine-independent software" [Ref. 8]. The uniqueness of the Mobile Programming System is that unlike most approaches to portability through macro processors, it does not require an existing version of

itself. The system can be bootstrapped on most contemporary computing machines starting with the simplified translator called SIMCMP.

1. SIMCMP

SIMCMP (simple compiler) is a simple macro processor used to translate any source language which can be defined by macros with single character parameters. The SIMCMP algorithm consists of two distinct phases, macro definition and macro expansion. During the macro definition phase, all macros are read-in using macro names or "templates" as the start of the definition and an end-of-line flag to delimit the definition. Once the definition phase is complete, expansion begins. In the expansion phase, each line of the source code is read. The source statement is matched with the predefined templates. If a match is found, the corresponding macro code body is punched out and the translator continues. If no template match is found for a line of source code, the code is treated as a statement in the base language. Reading a blank line from the source code terminates the process.

2. STAGE2

The second level of the bootstrapping process is STAGE2, consisting of a more powerful processor written in a language capable of being translated by SIMCMP. The purpose of STAGE2 is to provide a means for implementing machine independent software. Software support of STAGE2 has been kept minimal and, as Waite [Ref. 8] pointed out, only five basic functions are required of the supporting machine.

1. read one line
2. detect end-of-file
3. write one line
4. write end-of-file
5. rewind

STAGE2 employs a scanner to recognize macro calls and isolate parameter strings. Template matching is used for recognition of macro calls. Where SIMCMP was a very

simple processor, STAGE2 is a more powerful and flexible processor providing the user many functions. The macro facilities of STAGE2 are similar to more powerful macro assemblers, and will be reviewed below for completeness.

a. Storage

STAGE2 provides three levels of storage: parameter storage, associative memory, and up to nine input/output files.

Macros have storage space for nine parameters, with parameter storage local to the macro body. That is, the parameters are available only to the code body of the macro in which they were declared. Because nested macro calls are allowed, parameter values are saved during these calls. At the end of the code body, however, all parameters associated with that macro call are lost and not retrievable.

The associative memory is similar to the COMMON statement in FORTRAN programming. That is, the associative memory area is global to the entire set of macro calls and provides a means for transferring information between macros.

"By suitable processor functions the user can direct output generated by STAGE2 to any file on which writing is allowed. The input may be switched to any file on which reading is allowed" [Ref 8]. Multi-pass operations are possible since the input/output files can be used for temporary storage of intermediate text.

b. Arithmetic Expression Evaluation

Addition, subtraction, multiplication, and division can be performed by STAGE2. All arithmetic evaluations must be of integer type. That is, the operands must be either integers or symbols, where each symbol has an associated integer value stored in associative memory.

c. Conditional and Loop Operations

String equality and the relative magnitude of two expressions can be tested. As a result, an option is

available where the ability to skip a predetermined amount of code is available. The code body of the calling macro may be exited on this type of conditional branch. Looping within a macro call is also available. Similar in structure to a FORTRAN DO-Loop, the calling macros code body may be duplicated more than once.

As pointed out earlier, the Mobile Programming System can be bootstrapped on a target machine without an existing version of itself. SIMCMP is defined in FORTRAN because of the high availability of FORTRAN compilers. However, if a FORTRAN compiler is not available, SIMCMP is easily coded in assembly language on most computers. STAGE2 is written in FLUE (First Language Under Bootstrap) and can be compiled with SIMCMP. After STAGE2 has been implemented in its simplest form, it is "Bootstrapped" until the desired level is reached. As Poole and Waite [Ref. 1] point out, the resulting software is, therefore, tailored "to the particular problem at hand" and its complexity is extended only "as the need arises."

B. LIMP

LIMP (Language Independent Macro Processor) [Ref. 7] is presently used to process text for eight different compilers. LIMP embodies the concept of a macro processor which is independent of assembly language. Based upon the idea of string manipulation with parameter substitution, it has the capacity to allow the programmer to alter the behavior of a specific compiler without redefining the entire language.

Somewhat similar to the Mobile Programming System in operation, LIMP provides a much broader means of macro definition. Although still referred to as a template, the line which introduces the macro definition may be any arbitrary string of characters. A special parameter symbol is used to indicate the positions of parameters. Waite [Ref. 9] suggests "this approach frees the system from any

arbitrary format restrictions of a particular language - templates can be written in a form suited to the language at hand." Any code line which cannot be matched to a template is copied directly to the output source without change.

1. Parameters

A maximum of nine formal parameters may be specified within the macro code body. They are specified by their relative position in the macro definition template using the integers one through nine. Unlike the usual substitution of actual for formal parameters, LIMP allows several conversion types by indicating the desired subscript on the relative address of the parameter. The conversion digits describe the ways in which the actual parameter string may be used in each substitution. When more than nine formal parameters are needed, a private tree may be used to store intermediate results, as described below.

2. Trees

Three types of trees are available in LIMP: private tree, symbol tree and template tree. As mentioned earlier, private trees may be used to store intermediate results when an excess of nine parameter strings are needed. Except for the macro code body, which is converted to list structure when defined, all input source code is considered in string form. It is, therefore, not possible to use string operations to make changes within the code body. If the user foresees the need to alter the code body of a macro, the code body can be stored in the private tree as a string. It is then possible to redefine this string as the code body of any template.

The symbol tree is used to store symbols which are either created by parameter calls or direct declaration by the user. In conjunction with this, any symbols added to the tree are automatically given the current value of a built-in location counter or explicitly assigned by the user.

The template tree contains the templates of all

defined macros. When an input line is matched to a template, actual parameter strings are created and the associated macro code body is duplicated. The user may reenter the definition phase by using a special command to define another template and add it to the template tree. A blank line after the macro definition causes the system to return to the expansion phase.

The LIMP system does, to some extent, solve the problems of languages being tailored to fit target compilers. It allows flexibility in software which is critical to the problems of transferability. W. M. Waite suggests that, with planned extensions, LIMP could far surpass the role of present macro processors and "would have the power of a syntax directed compiler" [Ref. 9].

Both the Mobile Programming System and LIMP have enjoyed some degree of success. The Mobile Programming System has been implemented on eight different machines with a maximum of two man weeks required for implementation. As Poole and Waite point out, "it is not even necessary to have access to a running version of the system on another machine; a listing, tape or BCD deck is sufficient" [Ref. 1]. LIMP, on the other hand, was not designed with the same goals in mind. The degree of independence of LIMP is directly related to the availability of a WISP compiler on the target machine. WISP is currently available on the IBM 7094, 7040, GE 265, ENGLISH ELECTRIC KDF9, ELLIOT 803, EDSAC 2 and ATLAS 2 computer systems. Once implemented, LIMP is capable of processing text for eight different compilers. As a result, text acceptable to LIMP attains some degree of machine independence.

While the use of macro processors is a valid approach to portability, the development of standardized high-level language compilers opens the door to programming in a high-level language to attain a degree of portability. Software written in high-level languages have been reasonably successful in the field of transferrability.

III. HIGH-LEVEL LANGUAGE PROGRAMMING

The use of high-level languages is evident in many attempts to enhance portability. The macro processors mentioned previously can themselves be highly portable systems if written in a widely available high level language. If this approach is not taken, the job of transferring the system deteriorates to one of reprogramming in assembly language. The alleviation is to code the applications programs by coding directly in universally available high-level languages.

A. STANDARDIZATION

The need for standardization of programming languages is evident. "This opinion is reflected by the existence of standards committees such as ANSI (American National Standards Institute) which have been organized for most popular languages" [Ref. 5]. Industry is very much concerned with maintaining a degree of standardization. "Presently, there are COBOL precompilers that enforce management selected language usage constraints. Information Management Inc., for instance, has MAGIC Standards Enforcer which notes use of management-prescribed COBOL verbs. Arkay Computer Applications' MECCA flags about 200 management selected violations" [Ref. 5]. These standard enforcement packages are largely oriented toward maintaining uniformity of the listings for the sake of readability, but they can flag the use of any selected reserved words. In order to standardize programming languages, in 1969 the UNITED STATES NAVY issued OPNAVINST 10462.8. This instruction explains the Navy's need for standardization and sets forth well documented standards for the COBOL, FORTRAN, and JOVIAL programming languages. Standardization allows the coding of applications programs within the constraints of a programming language. The implementation of that program on

any target machine is dependent on compiler availability and a minimum consideration of machine-dependent aspects, such as word size, I/O interfaces, character sets, and peripherals. The availability of standardized high-level language compilers is a problem which is rapidly decreasing. As early as 1966, the Navy attempted standardization by issuing SECNAVINST 10462.8 which demands that all new computer acquisitions must specify COBOL and FORTRAN compilers as a mandatory requirement. The Navy realizes that certain high-level languages are better for different user requirements, and, while it limits and standardizes the allowed high level languages, it provides a wide base for running its software.

Standardization is lost when a language is extended to meet certain specialized needs. It seems that even the most powerful high level language compilers are altered to provide extra features deemed necessary by implementors of the compiler. These features are added at the expense of the transferability of programs coded for processing by the "improved" compilers. In time, however, even these extended languages may become standardized. ANSI FORTRAN, for example, is an extension of the original FORTRAN IV language. If extended languages are implemented, their compilers can be written to scan source code for factors which influence portability. "Whenever possible, compilers should flag usage of non-standard language features that they implement. WATFOR ... is an excellent example of fulfilling this function"[Ref. 5].

One successful method of extending a language is to write a preprocessor which converts the extended language into a standardized language. MORTRAN, a language developed at Stanford University, is an example of high-level language extension without significant loss of portability. Written in ANSI standard FORTRAN IV, the MORTRAN processor can be implemented on a wide variety of machines. The MORTRAN processor translates the MORTRAN input text directly into

FORTIRAN statements which can be compiled by the user's FORTRAN compiler. MORTRAN's features include:

1. free field format
2. alphanumeric statement labels
3. comments inserted anywhere in the text
4. multiple assignments
5. implicit looping and block structure
6. statements such as FOR-BY-TO, WHILE, UNTIL, and IF-THEN-ELSE
7. user defined macro-instructions
8. blocks of normal FORTRAN statements inserted in the text

These extensions afford the FORTRAN user a powerful language which retains the qualities of portability inherent in a standardized language such as FORTRAN.

B. STRUCTURED PROGRAMMING

Programming in a standardized, widely available high-level language offers more than just a multiplicity of compilers. The programmer has the opportunity to enhance the transferability of software by using the techniques of structured programming. Structured programming can be thought of as "the goal of evolving a program so that its final structure can be presented or described as a hierarchy of tasks or blocks" [Ref. 10]. Because those areas of computer software which retain machine dependencies must be altered when a transfer of software is to be affected, it is beneficial to the personnel performing the transfer to easily identify those dependencies. Structured, or modular programming, makes it "possible to study the system one module at a time" [Ref. 5] and therefore increases the ease of comprehension of the system. This type of modular programming can be used to segregate those areas in the software which may contain machine dependencies. Knowledge of the entire system or unrelated modules is, therefore, not required to accomplish the task of making the needed changes.

FORTRAN is an excellent example of a high-level language which can be easily modularized. "The ability to divide machine/system dependent and independent attributes into separate entities via sub-programs greatly enhances the transferability of programs" [Ref. 5]. The compiler for INTEL's PL/M language was written in ANSI standard FORTRAN [Ref. 11]. Because the compiler was written using the concepts of structured programming, machine dependencies were easily isolated from the remainder of the system. As a direct result of the modular design and standardized base language, PL/M has been easily implemented on eight different machines with little or no difficulty. Input/output file names are contained within two subroutines and can easily be accessed for change. Once implemented on a specific computing machine, sections of the PL/M compiler may be accessed to increase efficiency. This has been demonstrated on the IBM System/360. A forty percent decrease in compile time was achieved by replacing the FORTRAN method of shifting with an assembly language shift and mask operation. This adjustment, plus the implementation of an assembly language input/output package and the use of the FORTRAN H optimizing compiler, allowed compilation of PL/M programs in less than half the time required by the first version. The total effort expended to facilitate these changes was twenty-four man hours.

The advantages of coding in a standardized language similar to FORTRAN are apparent. As Hamlet [Ref. 12] points out, "only schemes based on ANSI FORTRAN really move easily among different computers." As discussed earlier, the Mobile Programming System is written partly in ANSI FORTRAN. The portability of the entire system is somewhat dependent upon the ability to implement SIMCMP on the target computer. As a result of the variety of portable software which has been written, an ANSI FORTRAN based preprocessor/editor was written with the goal of achieving some degree of portability.

IV. PREDITOR

PREDITOR (PREprocessor/EDITOR) was implemented in ANSI FORTRAN using the concepts of structured programming. This was considered to be a viable approach to machine independent software. PREDITOR is a preprocessor/editor which is capable of processing a small super-set of FORTRAN, called FORTRAN+, and allows changes to existing programs with a simplified text editor.

As an implementation language, FORTRAN is deficient in some areas. For example, ANSI FORTRAN does not provide a string data type. Some aspects of FORTRAN, such as the COMMON statement for representing global variables, have caused many man hours of work when a program change is necessary. FORTRAN+ attempts to solve some of the inadequacies of character manipulation inherent in FORTRAN and make the alteration of large software packages an easier task. In addition, the text editor section of PREDITOR includes four useful editing functions.

1. insert a line of code
2. delete a line or sequence of code
3. copy a predefined sequence of code
4. sequence the source file

The above functions are invoked using a number of "control toggles," a patch file containing desired changes, and a user's sequenced source file.

A. INDEPENDENT ASPECTS

Because of the availability of standard FORTRAN compilers, PREDITOR offers a highly portable package. Structured programming and machine-independent character representation allows the data structures and internal logic to be transparent to the user and target machine. The modular design has also allowed for the isolation of input/output processing. This results in a flexible system

for user reassignment of peripheral devices, particularly in the program transfer stage. Word size varies widely from machine to machine, and thus must be considered an important factor in portability. PREDITOR uses machine word size to its advantage. By packing applicable input data, memory requirements are minimized. This allows existing software to be preprocessed for use on target machines with different word lengths while minimizing machine storage requirements.

B. DEPENDENT ASPECTS

PREDITOR contains some machine dependencies. The data which PREDITOR must access during execution is packed according to host machine word size. If PREDITOR is transferred to another machine, this data must be repacked according to the new machine word size. PREDITOR itself can be used to process the appropriate statements and provide a copy of the reformatted data. The syntax tables which are vital to the compiler portion of PREDITOR are also machine dependent and can be reconstructed for the host machine.

Special characters which trigger PREDITOR functions may conflict with predefined uses of those characters in existing target machines. For example, the dollar sign is used as a comment indicator in the XDS Sigma 7. The reserved dollar character can easily be changed in any implementation.

C. METHCDS OF TRANSFER

Prior to transfer to a new environment, certain dependencies must be accounted for. Word size dependent syntax tables and packed output data must be replaced with tables and data generated for the target machine.

Input/output files presently reflect the file numbers of the IBM System 360. These file numbers may be redefined to meet user requirements or existing system file definitions. All areas which may need to be altered are clearly marked in the source listing of PREDITOR.

If a conflict involving the use of special characters

which trigger PREDITOR functions exists, PREDITOR must be altered. An alternate character must be defined and implemented in the system.

D. OPERATING PROCEDURES

PREDITOR is a one pass system. During the pass, the source program is scanned and all improperly formed statements are detected. A listing of the source program can be obtained and errors are listed by line number. Error messages take the form:

```
***** ERROR(x) NEAR LINE y
***** Brief explanation of error
```

The number x corresponds to the error number and y is the line number where the error occurred. The line which immediately follows gives a brief explanation of the particular error message. See Appendix C for a complete list of error messages.

File numbers used in PREDITOR, along with the corresponding device types are listed in Appendix D. These I/O file numbers may be set prior to execution or during execution by using available control switches.

PREDITOR begins by reading a record from the input file. The PREDITOR processor translates the FORTRAN+ input text directly to FORTRAN statements, which are then compiled by the user's FORTRAN compiler. \$\$ Control switches are usually set at this time by the user. If not, control switches are set to default values. If the edit mode is not selected, the source file will be preprocessed as is. If editing is desired, GOFERS will merge the patch deck and the source file and process it normally. Figures 1 and 2 show typical patch deck and source file composition. When the nine cards are read, the operation is complete. The resulting merged file is then ready to be compiled from the temporary storage area.

\$\$ Control Switches

(these consist of either control toggles
or control switches and are used as desired.)

\$\$PATCH (Required)

\$\$ Statements

(These cards are used to delete or add code
to the source file.)

99999999 (nines card required.)

Figure 1

A Typical Preditor Patch File

\$\$xx Statements (if desired)

(These are used in conjunction with
the \$ COPY command)

\$\$ SOURCE (Required)

\$ Command Statements

(These include \$ OUTPUT, \$ COPY and
\$ APPEND VERSE.)

(nines card required)

99999999

Figure 2

A Typical Preditor Source File

E. \$\$ STATEMENTS

The \$\$ Statements are used to initialize PREDICTOR and alter the predefined states of control toggles. \$\$ SOURCE marks the beginning of the source file. All statements following the \$\$ SOURCE are considered as part of the source file until a "nines card" (99999999 in columns 73-80) is detected.

The \$\$ PATCH indicates the beginning of the patch deck. The patch deck defines additions and deletions to the source file. All statements following the \$\$ PATCH are considered as part of the patch deck until a nines card (99999999 in columns 1-8) is detected. Patch deck cards are of two forms:

1. insert a card of code
xxxxxxx CODE BODY
2. delete a body of code
xxxxxxx*yyyyyyyy

The xxxxxxxx field is in columns 1 - 8 and indicates that the CODE BODY (columns 73-80) is to be inserted prior to a source file record of equal or greater sequence number. If a star (*) appears in column nine, the records to be deleted begin with sequence number xxxxxxxx and end with sequence number yyyyyyyy inclusive.

\$\$xx statements are used to define a portion of text which is to be copied into the executable source file. The dollar signs (\$\$) are placed in columns 1 and 2 with the xx field in columns 3 and 4. The xx field is any integer between 1 and 99 and represents all text which follows until another dollar sign is found in column 1. This text may be copied into the new source file with the call \$ COPY xx. This allows the user to make changes to statements such as FORTRAN COMMON, which may be used frequently throughout a program. To facilitate the change, the user need only make the change in the definition stage as mentioned above. All \$\$xx statements and their associated text are physically placed directly before the \$\$ SOURCE card.

Other control switches take two forms: control toggles and control parameters. Toggles take on the values 1 and 0 (true or false), while parameters take on appropriate values.

Control switches are specified by typing a \$\$ in columns 1 and 2, and a switch name starting in column 3 (only the first character of the switch name is significant). The switch name is followed by an equal sign (=) and an integer value. Control switches are listed in Appendix B.

E. \$ COMMAND STATEMENTS

The \$ Command Statements are used freely within the source file after the \$\$ SOURCE card. These statements are part of the processor portion of PREDITOR. PREDITOR translates \$ Command Statements to standard FCRTAN subroutine calls.

1. \$ OUTPUT

The \$ OUTPUT allows output of strings and integers. The use of the concatenation feature within the COUTPUT statement provides for a continuous output of strings and integers. A dollar sign (\$) is placed in column 1 of the source file, followed by at least one blank, with the remainder of the statement being free format. The COUTPUT statement takes the following form:

\$ OUTPUT()

All information to be written out is contained between the two parentheses. The statement must terminate with a right parenthesis prior to column 73.

a. Strings

Strings are delimited by the single quote ('). All characters which appear between two quotes are considered part of the string. The use of two adjacent quote marks inside the string results in the output of a single quote. Although the maximum length of text that one OUTPUT statement can process is 64 characters, the output of a longer text can be performed using the concatenation

feature in conjunction with as many OUTPUT statements as are needed. Concatenation may be used to suppress dumping of the output buffer. This allows strings or integers which follow to be placed in the current buffer as shown in the following example.

```
$ OUTPUT('THIS IS A TEST '///  
$ OUTPUT(///  
'STRING')
```

The output would appear as THIS IS A TEST STRING.

b. Integers

Integers may be represented as variable names or constants. Four bases are available: binary, octal, decimal, or hexadecimal. A variable field width is also available with the option of blanks or zeros to fill out the field. To output an integer variable or constant, the statement must be of the form:

```
(x) nnnnnn : y
```

The value x in parentheses is optional and specifies a field width. If none is specified, a default width of 5 is used. If a negative value is given, the field is padded with blanks instead of zeros. The nnnnnn field is the variable or constant field, while the y represents the radix desired and is also optional, but defaults to base 10 if none is specified. All three components may be either integer values or variables with integer values. The negative concatenation option (///
/-/) can be used effectively in conjunction with the field width to produce a desired output. A ///
/-/ will cause all leading blanks in a field width to be deleted. This allows integers to be left-justified when concatenated with strings or other numbers as shown in the following example.

```
$ OUTPUT('THE VALUE OF K IS '///  
/-/(-10)K:10)
```

The output would appear as THE VALUE OF K IS 5 where K has the value 5.

2. \$_COPY

The \$COPY statement causes the text associated with the statement to be inserted at that point in the source

code. The statement must take the form:

\$ COPY xx

The dollar sign (\$) is placed in column 1, followed by at least one blank, with the remainder of the statement free format. The number xx is any integer between 1 and 99.

3. \$ APPEND VERSE

The \$ APPEND VERSE has the form:

\$ APPEND VERSE

The dollar sign must again be placed in column 1, followed by at least one blank, with the remainder of the statement format free. This causes the vector VERSE, which contains the packed output data, to be copied into the new source file at this point. The associated FORTRAN equivalence statements will be built at this time.

V. SUMMARY

Within the areas discussed, each effort made to increase the portability of software has offered some distinct advantages. Both the Mobile Programming System and LIMP are macro processing systems which were developed for constructing machine independent software, but attack the problem with different concepts in mind. While the Mobile Programming System was designed to be transferred with the software, LIMP must meet this goal by processing programs at those installations which support the language WISP. The Mobile Programming System is capable of being bootstrapped on most computing machines to a level which best meets the needs of the software to be processed. LIMP, on the other hand, was designed to translate the software into one of eight different languages. Although LIMP itself is bound to those systems which support WISP, the resulting software may be transferred to any machine which supports one of those eight languages.

The use of high-level language programming has also made a significant contribution to software portability. Standardization of programming languages and the attempt by many software manufacturers to produce a product which is easily understood and modified, has made transferability of software a much easier problem to solve. All methods discussed have experienced a degree of success. There is, however, another question which must be considered when the subject of portability arises. That is, what is the efficiency of the final product. Because most systems designed for transferability are coded with the concept of structured programming in mind, critical routines can be singled out and recoded in assembly language to increase efficiency. Although there are currently no comparative studies of efficiency of the macro processor systems

mentioned, Waite [Ref. 6] suggests that the "effort involved in this recoding is miniscule compared with that involved in hand-coding the entire processor." The success of this methodology is evident in the implementation of PL/M on the IBM System 360. With a minimum amount of alteration, efficiency was increased by approximately forty percent.

In the final analysis, the efficiency of the systems and methodologies used will be determined by the degree of development in those areas. The mobility associated with macro processors and high-level languages, however, has been demonstrated as a viable tool for implementation of programming languages on a variety of computing machines.

APPENDIX A: FORMAL DEFINITION OF PREDITOR

1. <PROGRAM> ::= <STATEMENT LIST>
2. <STATEMENT LIST> ::= <STATEMENT>
3. | <STATEMENT LIST><STATEMENT>
4. <STATEMENT> ::= <OUTPUT STATEMENT>
5. | <APPEND STATEMENT>
6. | <COPY STATEMENT>
7. <OUTPUT STATEMENT> ::= <OUTPUT HEAD>)
8. <APPEND STATEMENT> ::= <APPEND><SUBJECT>
9. <COPY STATEMENT> ::= <COPY><NUMBER>
10. <OUTPUT HEAD> ::= <OUTPUT> (
11. | <OUTPUT HEAD><STRING>
12. | <OUTPUT HEAD><CONCAT>
13. | <OUTPUT HEAD><NCONCAT>
14. | <OUTPUT HEAD><CONOUT CALL>
15. <CONOUT CALL> ::= <CONOUT STATEMENT>
16. | <CONOUT CALL><RADIX>
17. <CONOUT STATEMENT> ::= <IDENTIFIER>
18. | <NUMBER>
19. | <FIELD WIDTH><IDENTIFIER>
20. | <FIELD WIDTH><NUMBER>
21. <FIELD WIDTH> ::= (<NUMBER>)
22. <RADIX> ::= <RADIX HEAD><IDENTIFIER>
23. | <RADIX HEAD><NUMBER>
24. <RADIX HEAD> ::= :
25. <COPY> ::= COPY
26. <OUTPUT> ::= OUTPUT
27. <APPEND> ::= APPEND
28. <SUBJECT> ::= VERSE

APPENDIX B: PREDITOR CONTROL TOGGLES

<u>Switch Name</u>	<u>Use</u>	<u>Default</u>
\$\$APATCH=n	The patch deck input file number = n	1
\$\$BSEQUENCE=n	Begin:sequence numbering at n	25
\$\$DELTA=n	Sequence number increment = n	25
\$\$EMARGIN=n	Set the output line width to n characters.	120
\$\$HWDSIZE=n	Host machine wordsize = n	32
\$\$INPUT=n	Switch to file n for subsequent input ,(see file numbering).	1
\$\$MERGE	Edit mode: T or F	0
\$\$OUTPUT=n	Write subsequent output lines to file n	1
\$\$PRINT	Output lines to the printer: T or F	1
\$\$RMARGIN=n	Logical record size = n	80
\$\$SEQUENCE	Resequenece the source file: T or F	0
\$\$WORDSIZE=n	Target machine wordsize = n	32

NOTE: All input files are maximum of 80 characters, and the output files cannot exceed 120 characters.

APPENDIX C: ERROR MESSAGES

Error Number	Message
1	Illegal symbol pair. The symbols printed cannot appear in a valid statement. This error may have been caused by a previous error in the program.
2	Parse stack overflow. Simplify the statement or declare a larger parse stack and recompile.
3	Table overflow. Error may be result of too many variables in use at one time. Either simplify the OUTPUT statement or increase the varc size and reccmpile.
4	Improperly formed statement. The statement above is improperly formed. Check formal BNF for definition and usage.
5	<p>Inccrrect format for concatenation used. Proper format is:</p> <p>// : Append to the current output buffer.</p> <p>/-/ : Append to the current output buffer and delete the leading blanks.</p>
6	String length exceeds allowable limits. Use shorter string length combined with concatenation.
7	Variable exceeds FORTRAN limits. Variable listed below is greater than six characters in length. Rename the variable.
8	<p>A \$\$CONTROL statement is improperly formed. Check all CONTROL statements to insure proper format.</p> <p>\$\$ SOURCE</p> <p>\$\$ PATCH</p> <p>\$\$CCNTROL TOGGLE</p>

APPENDIX D: FILE DEFINITIONS

INPUT							OUTPUT		
<u>NUMBER</u>	<u>DEVICE</u>	<u>UNIT</u>	*	<u>NUMBER</u>	<u>DEVICE</u>	<u>UNIT</u>			
1	CARDRDR	5	*	1	PRINTER	6			
2	TELETYPE	4	*	2	TELETYPE	4			
3	PAPTAPE	6	*	3	PUNCH	7			
4	MAGTAPE	7	*	4	MAGTAPE	8			
5	DSKFILE	8	*	5	DSKFILE	9			
6	DSKFILE	9	*	6	DSKFILE	10			
7	AVAILABLE	10	*	7	PAPTAPE	11			

NOTE: Unit numbers may be altered to fit user requirements by accessing subroutines RDTEXT and WRITEL. These are the only references to these unit numbers.


```

C $10
C $20
C $30
C $1
$ $ $ $
$
$

*****
** THIS IS A SAMPLE OF A SOURCE FILE PRIOR TO BEING **
** PROCESSED BY PREDITOR *****
*****
INTEGER IBUFF(80), OBUFF(120), IBP, CBP, OTRAN(64), MARGIN
COMMON /FILES/ IBUFF, OBUFF, IBP, OBP, OTRAN, MARGIN
INTEGER CONTRL(64), IFILE, OFILE
COMMON /CONTRL/ CONTRL, IFILE, OFILE
INTEGER /VERSE(500)
COMMON /TEXT/ VERSE
SOURCE
CCPY 10
CCPY 20
CCPY 30
MARGIN = 120
CONTRL(20) = 1
CONTRL(26) = 1
IFILE = CONTRL(20)
OFILE = CONTRL(26)
OUTPUT = ('THIS IS A TEST OF PREDITOR')
I = 4
J = I + J
K = J
OUTPUT ('THE VALUE OF K IS '/-/' (-5) K : 10)
STOP
END

```

```

00000005
00000055
00000105
00000155
00000205
00000255
00000305
00000320
00000335
00000355
00000405
00000455
00000505
00000555
00000605
00000655
00000705
00000755
00000805
00000855
00000905

```



```

$ $
SUBROUTINE WRITEL(NSPACE)
CCPY 10
CCPY 20
IF(OBP .EQ. 0) GO TO 999
NBLANK = 1
DO 5 I=1,OBP
  J = OBUFF(I)
  IF(J .NE. 1) NBLANK = I
  OBUFF(I) = OTRAN(J)
5 CONTINUE
OBP = IMIN(120,NBLANK)
GC TO (10),UFILE
WRITE(6,1002) (OBUFF(I),I=1,OBP)
10 IF(NSPACE .EQ. 0) GO TO 20
11 DO 15 I=1,NSPACE
  WRITE(6,1000)
15 CONTINUE
20 OBP = 0
999 RETURN
1001 FORMAT(1H )
1002 END

```

```

SUBROUTINE PAD(LC,CHR,I)
INTEGER LC,CHR,I,T(20)
J = IMIN(I,20)
DO 10 K=1,J
  T(K) = CHR
10 CONTINUE
CALL FORM(LC,T,1,J,20,.FALSE.)
RETURN
END

```

00000955
00001005
00001055
00001105
00001155
00001205
00001255
00001305
00001355
00001405
00001455
00001505
00001555
00001605
00001655
00001705
00001755
00001805
00001855
00001905
00001955
00002005

00002055
00002105
00002155
00002205
00002255
00002305
00002355
00002405
00002455


```

SUBROUTINE CONOUT(LC,K,N,BASE)
INTEGER K,N,BASE,T(20),LC
COPY 10
LOGICAL ZSUP,AFLAG
NP=N
ZSUP = K .LT. 0
KP = IMIN(IABS(K),19)
DO 10 I=1,KP
  T(I) = 1
10 CONTINUE
  IP = KP + 1
  DO 20 I=1,KP
    T(IP-I) = MOD(NP,BASE) + 2
    NP = NP/BASE
    IF(ZSUP .AND. (NP .EQ. 0)) GO TO 30
20 CONTINUE
  AFLAG = .FALSE.
  IF(.NOT. AFLAG) GO TO 35
  IF(BASE .EQ. 8) GO TO 40
  IF(BASE .EQ. 2) GO TO 45
  IF(BASE .NE. 16) GO TO 50
  KP = KP + 1
  T(KP) = 19
  GO TO 50 + 1
40 KP = KP + 1
  T(KP) = 28
  GO TO 50 + 1
45 KP = KP + 1
  T(KP) = 13
50 CALL FORM(LC,T,1,KP,20,.FALSE.)
  RETURN
END

```

```

00002505
00002555
00002605
00002655
00002705
00002755
00002805
00002855
00002905
00002955
00003005
00003055
00003105
00003155
00003205
00003255
00003305
00003355
00003405
00003455
00003505
00003555
00003605
00003655
00003705
00003755
00003805
00003855
00003905
00003955
00004005
00004055

```



```

$ $
SLBRoutine FORM(LC,CHARS,START,FINISH,LENGTH,UNPAC)
COPY 10
INTEGER UNPAC
LOGICAL UNPAC
DUMLN = LENGTH
J = START
I = LC + 1
GC TO (100,200,300),I
CALL WRITEL(0)
100 IF(UNPAC) GO TO 250
200 IF(J.GT.FINISH) GO TO 999
OBP = OBP + 1
OBUFF(OBP) = CHARS(J)
J = J + 1
IF (OBP.GE. MARGIN) GO TO 100
GC TO 200
250 N = CTRL(19)/6
260 IF(N.LE. DUMLN) GO TO 270
N = MOD(DUMLN,N)
270 K = OBP + N
IF (K.GT. MARGIN) CALL WRITEL(0)
OBP = OBP + N
CALL UNPACK(CHARS(J),N)
DUMLN = DUMLN - N
J = J + 1
IF(J.GT.FINISH) GO TO 999
GC TO 260
300 IF(J.GT.FINISH) GO TO 999
IF(CHARS(J).NE. 1) GO TO 200
J = J + 1
GC TO 300
999 RETURN
END

```

```

00004105
00004155
00004205
00004255
00004305
00004355
00004405
00004455
00004505
00004555
00004605
00004655
00004705
00004755
00004805
00004855
00004905
00004955
00005005
00005055
00005105
00005155
00005205
00005255
00005305
00005355
00005405
00005455
00005505
00005555
00005605
00005655
00005705
00005755

```


00005805
00005855
00005905
00005955
00006005
00006055
00006105
00006155
00006205
00006255
00006305
00006355
00006405
00006455
00006505
00006555
00006605
00006655
00006705
00006755
00006805
00006855
00006905

00006955
00007005
00007055
00007105

00007155
00007205
00007255
00007305

00007355
00007405
00007455
00007505
00007555
00007605
00007655

```

SUBROUTINE UNPACK(WORD,I)
  CCOPY 10
  CCOPY 20
  INTEGER SHL,SHR,SHLCNT,WORD,I
  SHLCNT = 0
  DO 100 J=1,I
    K = SHL(WORD,CONTRL(19)-6*J)
    IF(K.GE.0) GO TO 80
    IF(K.GE.0).OR.(SHLCNT.GE.6) GO TO 60
    SHLCNT = SHLCNT + 1
    K = SHL(WORD,CONTRL(19)+SHLCNT-6*J)
    GO TO 50
    K = SHR(K,CONTRL(19)+SHLCNT-6)
    DO 70 L=1,SHLCNT
      K = K + 2**(6-L)
    CONTINUE
    SHLCNT = 0
    GO TO 90
    K = SHR(K,CONTRL(19)-6)
    OBUFF(OBP-J+1) = K
  CONTINUE
  RETURN
END

```

```

INTEGER FUNCTION SHL(I,J)
  SHL = I*(2**J)
  RETURN
END

```

```

INTEGER FUNCTION SHR(I,J)
  SHR = I/(2**J)
  RETURN
END

```

```

FUNCTION IMIN(I,J)
  IF(I.LT.J) GO TO 10
  IMIN = J
  GO TO 999
  IMIN = I
  RETURN
END

```

10
999

00007705
00007755
00007805
00007855
00007905
00007955
00008005
00008055
00008105
00008155
00008205
00008255
00008305
00008355
99999999

\$

BLOCK DATA
CCPY 10
DATA IBUFF/60*0/,CBUFF/120*0/,IBP/81/,OBP/0/
DATA OTRAN/
1 1H,1H0,1H1,1H2,1H3,1H4,1H5,1H6,1H7,1H8,1H9,
2 1HA,1HB,1HC,1HD,1HE,1HF,1HG,1HH,1HI,1HJ,1HK,
3 1HL,1HM,1HN,1HO,1HP,1HC,1HR,1HS,1HT,1HU,1HV,
4 1HW,1HX,1HY,1HZ,
5 1H\$,1H=,1H.,1H/,1H(,1H),1H+,1H-,1H*,1H,,
1H<,1H>,1H:,1H;,12*0/
CCPY 20
DATA CONTRL/64*0/,IFILE/0/,OFILE/0/
CCPY 30
APPEND VERSE
END

\$

\$

\$

\$


```

C      SUBROUTINE WRITEL(NSPACE)
C      COPY 10
C      INTEGER I,IBUFF(80),OBUFF(120),IBP,CBP,OTRAN(64),MARGIN
C      COMMON /FILES/ IBUFF,OBUFF,IBP,CBP,OTRAN,MARGIN
C      COPY 20
C      INTEGER CONTRL(64),IFILE,OFILE
C      COMMON /CONTRL/ CONTRL,IFILE,OFILE
C      IF(CBP .EQ. 0) GO TO 999
C      NBLANK = I
C      DC 5 I=1,OBP
C      J = OBUFF(I)
C      IF(J .NE. 1) NBLANK = I
C      OBUFF(I) = OTRAN(J)
C      CONTINUE
C      OBP = IMIN(120,NBLANK)
C      GC TO (10),OFILE
C      10 WRITE(6,1002) (OBUFF(I),I=1,OBP)
C      11 IF(NSPACE .EQ. 0) GO TO 20
C      DC 15 I=1,NSPACE
C      WRITE(6,1001)
C      15 CONTINUE
C      20 OBP = 0
C      999 RETURN
C      1001 FORMAT(1H )
C      1002 END

```

00000850
00000900
00000905
00000955
00000950
00000105
00000155
00001000
00001050
00001100
00001150
00001200
00001250
00001300
00001350
00001400
00001450
00001500
00001550
00001600
00001650
00001700
00001750
00001800
00001850
00001900

```

SUBROUTINE PAD(LC,CHR,I)
INTEGER LC,CHR,I,t(20)
J = IMIN(I,20)
DC 10 K=1,J
I(K) = CHR
10 CONTINUE
CALL FORM(LC,T,1,J,20,.FALSE.)
RETURN
END

```

00001950
00002000
00002050
00002100
00002150
00002200
00002250
00002300
00002350


```

C
SUBROUTINE CONOUT(LC,K,N,BASE)
  INTEGER K,N,BASE,T(20),LC
  CCOPY 10
  INTEGER IBUFF(30),OBUFF(120),IBP,CBP,OTRAN(64),MARGIN
  COMMON /FILES/ IBUFF,OBUFF,IBP,OBP,OTRAN,MARGIN
  LOGICAL ZSUP,AFLAG
  NP = N
  ZSUP = K,LT: 0
  KP = IMIN(IABS(K),19)
  DO 10 I=1,KP
    T(I) = 1
  10 CONTINUE
  IP = KP + 1
  DO 20 I=1,KP
    T(IP-I) = MOD(NP,BASE) + 2
    NP = NP/BASE
    IF(ZSUP .AND. (NP .EQ. 0)) GO TO 30
  20 CONTINUE
  AFLAG = .FALSE.
  30 IF(.NOT. AFLAG) GO TO 35
  35 IF(BASE .EQ. 8) GO TO 40
  IF(BASE .EQ. 2) GO TO 45
  IF(BASE .NE. 16) GO TO 50
  KP = KP + 1
  T(KP) = 19
  GO TO 50
  40 KP = KP + 1
  T(KP) = 28
  GO TO 50
  45 KP = KP + 1
  T(KP) = 13
  50 CALL FORM(LC,T,1,KP,20,.FALSE.)
  RETURN
  END

```

```

00002400
00002450
00002500
00000005
00000055
00002550
00002600
00002650
00002700
00002750
00002800
00002850
00002900
00002950
00003000
00003050
00003100
00003150
00003200
00003250
00003300
00003350
00003400
00003450
00003500
00003550
00003600
00003650
00003700
00003750
00003800
00003850
00003900
00003950

```



```

C      SUBROUTINE FORM(LC,CHARS,START,FINISH,LENGTH,UNPAC)
C      COPY 10
C      INTEGER IBUFF(80),OBUFF(120),IBP,CBP,OTRAN(64),MARGIN
C      COMMON /FILES/ IBUFF,CBUFF,IBP,CBP,OTRAN,MARGIN
C      COPY 20
C      INTEGER CONTRL(64),IFILE,QFILE
C      COMMON /CONTRL/ CONTRL,IFILE,QFILE
C      INTEGER CHARS(LENGTH),LC,START,FINISH,DUMLEN
C      LOGICAL UNPAC
C      DUMLEN = LENGTH
C      J = START
C      I = LC + 1
C      GO TO (100,200,300),I
C      CALL WRITEL(0)
100  IF(UNPAC) GO TO 250
200  IF(J.GT.FINISH) GO TO 999
      OBP = OBP + 1
      OBUFF(OBP) = CHARS(J)
      J = J + 1
      IF (OBP.GE. MARGIN) GO TO 100
      GO TO 200
250  N = CONTRL(19)/6
260  IF(N.LE.DUMLEN) GO TO 270
270  N = MOD(DUMLEN,N)
      K = OBP + N
      IF (K.GT. MARGIN) CALL WRITEL(0)
      OBP = OBP + N
      CALL UNPACK(CHARS(J),N)
      DUMLEN = DUMLEN - N
      J = J + 1
      IF(J.GT.FINISH) GO TO 999
      GO TO 260
300  IF(J.GT.FINISH) GO TO 999
      IF(CHARS(J).NE.1) GO TO 200
      J = J + 1
      GO TO 300
995  RETURN
      END

```

```

00004000
00004050
00000005
00000055
00004100
00000105
00000155
00004150
00004200
00004250
00004300
00004350
00004400
00004450
00004500
00004550
00004600
00004650
00004700
00004750
00004800
00004850
00004900
00004950
00005000
00005050
00005100
00005150
00005200
00005250
00005300
00005350
00005400
00005450
00005500
00005550
00005600
00005650

```



```

C      SUBROUTINE UNPACK(WORD,I)
C      COPY 10
C      INTEGER IBUFF(80),OBUFF(120),IBP,OBP,OTRAN(64),MARGIN
C      COMMON /FILES/ IBUFF,OBUFF,IBP,OBP,OTRAN,MARGIN
C      COPY 20
C      INTEGER CCNTRL(64),IFILE,OFILE
C      COMMON /CNTRL/ CNTRL,IFILE,OFILE
C      INTEGER SHL,SHR,SHLCNT,WORD,I
C      SHLCNT = 0
C      DC 100 J=1,I
C      K = SHL(WORD,CNTRL(19)-6*J)
C      IF(K .GE. 0) GO TO 80
C      IF((K .GE. 0).OR.(SHLCNT.GE.6)) GO TO 60
C      SHLCNT = SHLCNT + 1
C      K = SHL(WORD,CNTRL(19)+SHLCNT-6*J)
C      GO TO 50
C      K = SHR(K,CNTRL(19)+SHLCNT-6)
C      DO 70 L=1,SHLCNT
C      K = K + 2**(6-L)
C      CONTINUE
C      SHLCNT = 0
C      GO TO 90
C      K = SHR(K,CNTRL(19)-6)
C      OBUFF(OBP-J+1) = K
C      CONTINUE
C      RETURN
C      END
C
C      INTEGER FUNCTION SHL(I,J)
C      SHL = I*(2**J)
C      RETURN
C      END
C
C      INTEGER FUNCTION SHR(I,J)
C      SHR = I/(2**J)
C      RETURN
C      END
C
C      FUNCTION IMIN(I,J)
C      IF(I .LT. J) GO TO 10
C      IMIN = J
C      GO TO 999
C      IMIN = I
C      999 RETURN
C      END

```

```

00005700
00005750
00000005
00000055
00005800
00000105
00000155
00005850
00005900
00005950
00006000
00006050
00006100
00006150
00006200
00006250
00006300
00006350
00006400
00006450
00006500
00006550
00006600
00006650
00006700
00006750
00006800

00006850
00006900
00006950
00007000

00007050
00007100
00007150
00007200

00007250
00007300
00007350
00007400
00007450
00007500
00007550

```


C	BLOCK DATA		00007600
	CCPY 10		00007650
	INTEGER /FILES/	IBUFF(80), OBUFF(120), IBP, CBP, OTRAN(64), MARGIN	00000005
	COMMON /COMMON /	IBUFF(80*0/), OBUFF(120*0/), IBP/81/, OBP/0/	00000055
	DATA IBUFF/80*0/		00007700
	DATA OTRAN/1H	1H0, 1H1, 1H2, 1H3, 1H4, 1H5, 1H6, 1H7, 1H8, 1H9,	00007750
1		1HA, 1HB, 1HC, 1HD, 1HE, 1HF, 1HG, 1HH, 1HI, 1HJ, 1HK,	00007800
2		1HL, 1HM, 1HN, 1HO, 1HP, 1HQ, 1HR, 1HS, 1HT, 1HU, 1HV,	00007850
3		1HW, 1HX, 1HY, 1HZ,	00007900
4		1H\$, 1H=, 1H., 1H/, 1H(, 1H), 1H+, 1H-, 1H', 1H*, 1H,,	00007950
5		1H<, 1H>, 1H:, 1H;, 12*0/	00008000
C	CCPY 20	CONTRL(64), IFILE, OFILE	00008050
	INTEGER /COMMON /	CONTRL/ CONTRL, IFILE, OFILE	00000105
	DATA CONTRL/64*0/	IFILE/O/, OFILE/O/	00000155
C	COPY 30	VERSE(500)	00008100
	INTEGER /COMMON /	TEXT/ VERSE	00008150
	APPEND VERSE/		00000205
C	DATA VERSE/		00000255
	*128212, 122956, 122956, 6096, 124865, 107585, 112464, 62751, 1693, 128212,		00008200
	*122964, 122956, 102476, 62416, 61518, 51023, 128208, 6220, 96272, 5777,		
	*505; 83841,		
	*478*0/		
	END		
			00008250

E R R O R M E S S A G E S

ERROR NUMBER	MESSAGE
1	ILLEGAL SYMBOL PAIR. THE SYMBOLS PRINTED CANNOT APPEAR IN A VALID STATEMENT. THIS ERROR MAY HAVE BEEN CAUSED BY A PREVIOUS ERROR IN THE PROGRAM.
2	PARSE STACK OVERFLOW. SIMPLIFY THE STATEMENT OR DECLARE A LARGER PARSE STACK AND RECOMPILE.
3	TABLE OVERFLOW. ERROR MAY BE RESULT OF TOO MANY VARIABLES IN USE AT ONE TIME. EITHER SIMPLIFY THE OUTPUT STATEMENT OR INCREASE THE VARC SIZE AND RECOMPILE.
4	IMPROPERLY FORMED STATEMENT. THE STATEMENT ABOVE IS IM-PROPERLY FORMED. CHECK FORMAL BNF FOR LANGUAGE DEFINITION AND USAGE.
5	INCORRECT FORMAT FOR CONCATENATION USED. PROPER FORMAT IS: // :APPEND TO THE CURRENT OUTPUT BUFFER /-- :APPEND TO THE CURRENT OUTPUT BUFFER DELETING THE LEADING BLANKS
6	STRING LENGTH EXCEEDS ALLOWABLE LIMITS. USE SHORTER STRING LENGTH COMBINED WITH CONCATENATION.
7	VARIABLE EXCEEDS FORTRAN LIMITS. VARIABLE LISTED BELOW IS GREATER THAN SIX CHARACTERS IN LENGTH. RENAME THE VARIABLE.
8	A \$\$CONTROL STATEMENT IS IMPROPERLY FORMED. CHECK ALL CONTROL STATEMENTS TO INSURE PROPER FORMAT. \$\$ SOURCE \$\$CONTROL TOGGLE

```

INTEGER MAXPAC, WDSIZE, LINENB
CCMMUN /INIT/ MAXPAC, WDSIZE, LINENB
INTEGER VCCPY(37), CUMLEN(37), COMVEC(500), CPTR,
1SCURCE(6), PCNT, COMFIN(37)
LCGICAL ERFLAG, SOURCE
CCMMON /COPYS/ VCCPY, CUMLEN, COMVEC, CPTR, SOURCE, PCNT,
1ERFLAG, SOURCE, COMFIN
1INTEGER VERLEN, START, FINISH, LENGTH, PAKCNT, VERSE(500),
1CCMMON /FORMS/ VERLEN, START, FINISH, LENGTH, PAKCNT,
1VERSE, CC
1INTEGER RADIX(8), VARB(8), FW(8)
CCMMON /CNCUT/ RADIX, VARB, FW
1INTEGER IBUFF(80), OBUFF(120), IBP, CBP, ITRAN(256),
1OTRAN(64), MARGIN
CCMMON /FILES/ IBUFF, OBUFF, IBP, OBP, ITRAN, OTRAN, MARGIN
1INTEGER V(215), VLOC(47), VINDX(13), CI(57), CITRI(1), PRIB(53),
1PRIB(53), HDTB(53), PRLEN(53), CONTC(53), LEFTC(1), LEFTI(30),
2CCNTT(1), TRIP(30), PRIND(47), NSY, NT, VLEN, VIL, CIW, CIL, NCITRI, PRIBL,
3PROTBL, HDTBL, PRLENL, CCNCL, LEFTCL, LEFTIL, CONTL, TRIPL, PRIL, PACK,
4TOKEN, IDENTV, NUMBV, STRV, CATV, NCATV, EOFIL
CCMMON /SYNTAX/ V, VLOC, VINDX, CI, CITRI, PRIB, PROTB, HDTB, PRLEN, CCNTC,
1LEFTC, LEFTI, CCNTI, TRIP, PRIND, NSY, NT, VLEN, VIL, CIW, CIL, NCITRI,
2PRIBL, PRIBL, HDTBL, PRLENL, CCNCL, LEFTCL, LEFTIL, CONTL, TRIPL, PRIL,
3PACK, IQEN, SP, PSTACK(75), MSTACK, PRMASK(5), MF, MPPI
1INTEGER SP, PSTACK(75), MSTACK, PRMASK(5), MF, MPPI
LCGICAL FAILSF, COMPI
CCMMON /STACKS/ SP, PSTACK, MSTACK, PRMASK, MP, MPPI, FAILSF,
1CCMPI
1INTEGER ACCLEN, ACCUM(99), NUMB, IDENT, EOFLAG, SPECL, STR, STYPE, TYPE,
1CNCAT, VAR(75), VARTOP, VARC(256), MVAR
CCMMON /SCAN/ ACCLEN, ACCUM, NUMB, IDENT, EOFLAG, SPECL, STR, STYPE, TYPE,
1CNCAT, VAR, VARTOP, VARC, MVAR
LCGICAL AFLAG, TOGSET
CCMMON /AFLAG/ AFLAG, TOGSET
1INTEGER CONTRL(64), IFIL, OFIL
CCMMON /CONTRL/ CONTRL, IFIL, OFIL
1INTEGER PABUFF(80), SOBUFF(80), PANUM(8), PASNUM(8), MERGE
LCGICAL INSERT, FAULT
CCMMON /MERGES/ PABUFF, SOBUFF, SONUM, PANUM, PASNUM, MERGE, INSERT,
1FAULT
1INTEGER EQUIV(10), EQPTR, NUMCTR, NUMLN, MAXLN, CBPCK
CCMMUN /EQUIVA/ EQPTR, NUMCTR, NUMLN, MAXLN, CBPCK
1INTEGER TEXT(138)
CCMMON /TEXTS/ TEXT
1INTEGER SEQ
LCGICAL SEQ
CCMMON /SEQNS/ SEQNS, SEQ

```


INTEGER GNC

THE FOLLOWING CONTROLS ARE DEFINED AS FOLLOWS. THEY ARE SET BY THE USER TO PROVIDE THE OPTIONS HE DESIRES.

OPTION CONTROL NUMBER

PATCH INPUT = I (12)
 SEQUENCE NUMBER START = I (13)
 SEQUENCE NUMBER INCREMENT = I (15)
 PRINT MARGIN = I (16)
 PRINT WDSIZE = I (19)
 INPUT FILE = I (20)
 LEFT MARGIN = I (23)
 MERGET = F OR F I (24)
 OUTPUT = F OR F I (26)
 PRINT MARGIN = I (27)
 RIGHT MARGIN = I (29)
 SEQUENCE = T OR F (30)
 TRIGGER = I (31)
 TARGET WORD SIZE = I (34)

CCCTRL(12) = 1
 CCCTRL(13) = 0
 CCCTRL(15) = 50
 CCCTRL(16) = 120
 CCCTRL(19) = 32
 CCCTRL(20) = 1
 CCCTRL(23) = 10
 CCCTRL(24) = 1
 CCCTRL(26) = 1
 CCCTRL(27) = 1
 CCCTRL(29) = 80
 CCCTRL(30) = 0
 CCCTRL(31) = 38
 CCCTRL(34) = 32
 AFLAG = .FALSE.
 DC 1 PRMASK(I) = 2** (I*8-8) - 1
 CCNTINUE
 DC 2 I=1,256
 ITRAN(I) = 1
 CCNTINUE
 DC 5 I=53,64
 OTRAN(I) = OTRAN(1)
 CCNTINUE
 DC 10 I=1,52
 J = OTRAN(I)
 J = ICON(J)



```

10 ITRAN(J) = I
   CCNTINUE
   IF (CTRL(27) .EQ. 0) GO TO 14
   OFILE = 1
   MARGIN = CTRL(29)
   CALL FORM(0,TEXT,1,9,42,.TRUE.)
   CALL WRITEL(1)
14 IBP = 81
15 I = GNC(0)
16 J = 0
   IF (IBUFF(1) .EQ. 999) GO TO 999
   IF (IBUFF(1) .NE. CTRL(31)) GO TO 13
   IF (IBUFF(2) .NE. CTRL(31)) GO TO 30
   IF (IBUFF(3) .LE. 11) .AND. (IBUFF(3) .GE. 2) GO TO 35
   CALL SETTOG CTRL(34)
   WDSIZE = WDSIZE/6
   MAXPAC = WDSIZE/6
   GO TO 14
35 CALL COPY
   GO TO 16
30 J = 38
13 IBUFF(1) = 14
   IF (CTRL(30) .EQ. 0) GO TO 18
   CALL SEGDK
18 IF (CTRL(27) .EQ. 0) GO TO 17
   MARGIN = CTRL(16)
   LINENB = LINENB + 1
   CALL CONOUT(0,5,LINENB,10)
   CALL PAD(1,1,3)
   CALL FORM(1,IBUFF,1,80,80,.FALSE.)
   OFILE = 1
   CALL WRITEL(0)
   MARGIN = CTRL(29)
17 IF (CTRL(26) .EQ. 1) GO TO 20
   CALL FORM(0,IBUFF,1,80,80,.FALSE.)
20 IF (J .NE. CTRL(31)) GO TO 14
   DC 25 I=1,3
   PSTACK(I) = 0
25 CCNTINUE
   PSTACK(4) = EOFIL
   SP = 4
   CALL SCAN
   CC = 0
   CALL CLCOP
   GO TO 16
999 STOP
   END

```



```

INTEGER FUNCTION GNC(IQ)
*****
** GNC GETS THE NEXT CHARACTER FROM THE INPUT STREAM. **
** IF THE EDIT MODE IS SELECTED, GNC DECIDES WHICH **
** FILE TO READ FROM AND PASSES ON THE PROPER BUFFER. **
*****
INTEGER MAXPAC, WDSIZE, LINENB
COMMON /INIT/ MAXPAC, WDSIZE, LINENB
INTEGER /IBUFF(80), CBUFF(120), IBP, OBP, ITRAN, OTRAN, MARGIN
COMMON /FILES/ IBUFF, CBUFF, IBP, OBP, ITRAN, OTRAN, MARGIN
INTEGER /CONTRL(64), IFILE, QFILE
COMMON /CONTRL/ CONTRL(64), IFILE, QFILE
INTEGER /PABUFF(80), SOBUFF(80), SCNUM(8), PANUM(8), PASNUM(8), MERGE
COMMON /MERGES/ PABUFF, SOBUFF, SCNUM, PANUM, PASNUM, MERGE, INSERT,
1 IF FAULT
LOGICAL AFLAG, TOGSET
COMMON /FLAGS/ AFLAG, TOGSET
LOGICAL /LINES, MERGIT
IF (IBP .LE. 72) GO TO 999
IF (.NOT. TOGSET) GO TO 340
IF (.CONTRL(24) .EQ. 0) GO TO 350
IF (.CONTRL(24) .EQ. 0) GO TO 100
IF IFILE = CONTRL(12)
CALL RTEXT(3)
CALL SEGNUM(1) CONTRL(24) = 0 GO TO 200
IF (INSERT .AND. FAULT) GO TO 200
IF (INSERT .TRUE.)
IF IFILE = CONTRL(20)
CALL RTEXT(2)
CALL SEGNUM(3)
IF (.NOT. NINES(2)) GO TO 200
IF IFILE = 999
RETURN
IF (.NOT. MERGIT(PANUM, SCNUM)) GO TO 300
CALL TO 400
GO TO 310
IF IFILE = SOBUFF(1)
GO TO 310
IF IFILE = .FALSE.
INSERT 400
GO TO 340
IF IFILE = CONTRL(12)
GO TO 351
IF IFILE = CONTRL(20)

```



```

351 CALL RTEXT(1)
    CALL SEGNUM(4)
    IF (.NOT. NINES(2)) GO TO 400
    IBUFF(1) = 999
    RETURN
400 IBP = 2
    GNC = 0
    RETURN
999 GNC = IBUFF(IBP)
    IBP = IBP + 1
    RETURN
END

```



```

SUBROUTINE RTEXT(L)
*****
** RTEXT READS FROM THE PROPER INPUT FILE AND OPAC **
** RTEXT READS FROM THE PROPER INPUT FILE AND PLACES *
** IT IN THE APPROPRIATE BUFFER. **
*****
INTEGER CTRL(64), IFILE, OFILE
COMMON /CTRL/ CTRL, IFILE, OFILE
INTEGER Ibuff(80), Cbuff(120), IBP, OBP, ITRAN(256), OTRAN(64), MARGIN
COMMON /FILES/ Ibuff, Ibuff, Obuff, IBP, Cbuff, ITRAN, OTRAN, MARGIN
INTEGER PABUFF(80), SOBUFF(80), SCONUM(8), PANUM(8), PASNUM(8), MERGE
LOGICAL INSERT, FAULT
COMMON /MERGES/ PABUFF, SOBUFF, SCONUM, PANUM, PASNUM, MERGE, INSERT,
1 FAULT
INTEGER BUFFER(80)
GO TO (10,50,60,70,80,90,100), IFILE
10 READ(5,1000) BUFFER
GO TO 115
50 READ(4,1000) BUFFER
GO TO 115
60 READ(6,1000) BUFFER
GO TO 115
70 READ(7,1000) BUFFER
GO TO 115
80 READ(8,1000) BUFFER
GO TO 115
90 READ(9,1000) BUFFER
GO TO 115
100 READ(10,1000) BUFFER
115 DC 210 I=1,80
J = BUFFER(I)
J = ICON(J)
GO TO (215,220,225), L
IBUFF(I) = ITRAN(J)
GO TO 210
SOBUFF(I) = ITRAN(J)
GO TO 210
PABUFF(I) = ITRAN(J)
CONTINUE
FCRMAT(80A1)
RETURN
END

```

CC
CC
CC
CC
CC


```

C C C C C C C C
SUBROUTINE WRITEL(NSPACE)
** *****
** WRITEL OUTPUTS THE CURRENT OUTPUT BUFFER TO THE
** INDICATED FILE. THE PARAMETER NSPACE WILL INDICATE**
** THE NUMBER OF BLANK LINES TO FOLLOW THE OUTPUT **
** BUFFER.*****
** *****
INTEGER I,IBUFF(80),CUBUFF(120),IBP,OBP,ITRAN(256),OTRAN(64),MARGIN
COMMON /FILES/ IBUFF,OBUFF,IBP,OBP,ITRAN,MARGIN
INTEGER /CTRL(64),IFILE,OFILE
COMMON /CTRL/ CTRL,IFILE,OFILE
IF(OBP.EQ.0) GO TO 999
NBLANK = 1
DO 5 I=1,OBP
  J=1,CUBUFF(I)
  IF(J.NE.1) NBLANK = I
  OBUFF(I) = OTRAN(J)
5 CONTINUE
OBP = IMIN(120,NBLANK)
GC TO (10,30,40,50,60,70,80),OFILE
10 WRITE(6,1002) (CUBUFF(I),I=1,OBP)
11 IF(NSPACE.EQ.0) GO TO 20
DO 15 I=1,NSPACE
  WRITE(6,1001)
15 CONTINUE
20 GC TO 20
30 WRITE(4,1000) (OBUFF(I),I=1,OBP)
40 WRITE(1,1000) (OBUFF(I),I=1,OBP)
50 WRITE(8,1000) (OBUFF(I),I=1,OBP)
60 WRITE(9,1000) (OBUFF(I),I=1,OBP)
70 WRITE(10,1000) (OBUFF(I),I=1,OBP)
80 WRITE(11,1000) (OBUFF(I),I=1,OBP)
90 GC TO 10
95 RETURN
1000 FFORMAT(80A1)
1001 FFORMAT(1H)
1002 FFORMAT(1H,120A1)
END

```



```

C
C
C
C
C
SUBROUTINE PAD(LC,CHR,I)
**
** *****
** PAD PLACES A SPECIFIED NUMBER OF DESIRED CHARACTERS*
** INTO THE OUTPUT BUFFER.*
** *****
**
INTEGER LC,CHR,I,T(20)
J = IMIN(I,20)
DO 10 K=1,J
    T(K) = CHR
10 CCNTINUE
CALL FORM(LC,T,1,J,20,.FALSE.)
RETURN
END

```



```

SUBROUTINE CONOUT(LC,K,N,BASE)
** ***** **
** CONOUT PLACES ANY INTEGER CONSTANT IN ANY OF THE **
** FOUR BASES AVAILABLE AND WITH A SPECIFIED FIELD **
** WIDTH LESS THAN EQUAL TO 19. THIS FIELD IS FILLED **
** WITH ZEROS OR BLANKS, DEPENDING ON THE SIGN OF K **
** ***** **
INTEGER K,N,BASE,T(20),LC
INTEGER Ibuff(80),OBUFF(120),IBP,OBP,ITRAN(256),OTRAN(64),MARGIN
COMMON /FILES/ Ibuff,OBUFF,IBP,OBP,ITRAN,OTRAN,MARGIN
INTEGER /TEXT(138)
COMMON /TEXTS/ TEXT
LOGICAL /AFLAG,TOGSET
COMMON /AFLAG,TOGSET
LOGICAL ZSUP
NP = N
ZSUP = K
LT = 0
KP = I MIN(IABS(K),19)
DO 10 I=1,KP
  T(I) = 1
10 CONTINUE + 1
  IP = KP
  DO 20 I=1,KP
    T(IP-I) = MOD(NP,BASE) + 2
    NP = NP/BASE
    IF(ZSUP .AND. (NP .EQ. 0)) GO TO 30
20 CONTINUE
  IF(.NOT. AFLAG) GO TO 35
  J = CBP + I + 1
  IF(J .LE. 71) GO TO 35
  CALL WRITEL(0,'/ - /')
  CALL FORM(Q,TEXT,I0,11,6,.TRUE.)
35 IF(BASE .EQ. 8) GO TO 40
  IF(BASE .EQ. 2) GO TO 45
  IF(BASE .NE. 16) GO TO 50
  KP = KP + 1
  T(KP) = 19
  GO TO 50
40 KP = KP + 1
  T(KP) = 28
  GO TO 50
45 KP = KP + 1
  T(KP) = 13
50 CALL FORM(LC,T,1,KP,20,.FALSE.)
  RETURN
END

```

CCCCCCCC

C


```

SUBROUTINE FORM(LC,CHARS,START,FINISH,LENGTH,UNPAC)
*****
** FORM UNPACKS DATA IF NECESSARY AND PLACES IT INTO **
** THE OUTPUT BUFFER.*****
*****
INTEGER IBUFF(80),CIBUFF(120),IBP,CBP,ITRAN(256),OTRAN(64),MARGIN
COMMON /FILES/ IBUFF,CIBUFF,IBP,CBP,ITRAN,OTRAN,MARGIN
COMMON /CCTRL/ CCTRL(64),IFILE,OFILE
COMMON /CONTRL/ CONTRL,IFILE,OFILE
INTEGER MAXPAC,WDSIZE,LINENB
COMMON /INIT/ MAXPAC,WDSIZE,LINENB
INTEGER CHARS(LENGTH),LC,START,FINISH,DUMLEN
LOGICAL UNPAC
DUMLEN = LENGTH
J = START
I = LC + 1
GO TO (100,200,300),I
CALL WRITEL(0)
100 IF(UNPAC) GO TO 250
200 IF(J.GT. FINISH) GO TO 999
CBP = CBP + 1
CIBUFF(CBP) = CHARS(J)
J = J + 1
IF (CBP .GE. MARGIN) GO TO 100
GO TO 200
N = CONTRL(19)/6
250 IF(N .LE. DUMLEN) GO TO 270
260 N = MOD(DUMLEN,N)
270 K = CBP + N
IF (K .GT. MARGIN) CALL WRITEL(0)
CBP = CBP + N
CALL UNPACK(CHARS(J),N)
DUMLEN = DUMLEN - N
J = J + 1
IF(J .GT. FINISH) GO TO 999
GO TO 260
300 IF(J .GT. FINISH) GO TO 999
IF(CHARS(J) .NE. 1) GO TO 200
J = J + 1
GO TO 300
999 RETURN
END

```



```

CCCCCCC
SUBROUTINE PRSYM(CC,SYM)
** PRSYM PRINTS THE APPROPRIATE SYMBOLS FOR ERROR **
** MESSAGES. **
** **
INTEGER V(215), VLCC(47), VINDX(13), C1(57), C1TRI(1), PRTB(53),
1PRTB(53), HDTB(53), PRLN(53), CONTC(53), LEFTC(1), LEFTI(30),
2CCNTI(1), TRIPI(30), PRLN(47), NSY, NT, VLEN, VIL, C1W, C1L, NC1TRI, PRTBL,
3PRTBL, HDTB, PRLN, CONCL, LEFTCL, LEFTIL, CONIL, TRIPL, PRIL, PACK,
4TCKEN, I, IDENTV, NUMBV, STRV, CATV, NCATV, EOCFILE
COMMON /SYNTAX/ V, VLCC, VINDX, C1, C1TRI, PRTB, HDTB, PRLN, CONTC,
1LEFTC, LEFTI, CONTI, TRIPI, PRIND, NSY, NT, VLEN, VIL, C1W, C1L, NC1TRI,
2PRTBL, PRTBL, HDTB, PRLN, CONCL, LEFTCL, LEFTIL, CONIL, TRIPL, PRIL,
3PACK, I, IDENTV, NUMBV, STRV, CATV, NCATV, EOCFILE
INTEGER CC, SYM, SHL, SHR, RIGHT
1INTEGER PBUFF(29)
K = VLCC(SYM+1)
IF(SYM.GT. NT) GO TO 100
L = V(K)
CALL FORM(CC,V,K+1,K+L,NSY+1,.FALSE.)
GO TO 999
100 L = RIGHT(K,15) - 1
K = SHR(K,15)
KP = 0
DO 200 I=1,K,PACK
L = L + 1
LP = V(L)
JP = PACK * 6
DO 200 J=1,JP-6
KP = KP + 1
IP = SHR(LP,JP)
PBUFF(KP) = RIGHT(IP,6) + 1
200 CONTINUE
CALL FORM(CC,PBUFF,1,K,30)
999 RETURN
END

```



```

SUBROUTINE UNPACK(WORD,I)
INTEGER IBUFF(80),OBUFF(120),IBP,OBP,ITRAN(256),OTRAN(64),MARGIN
COMMON /FILES/ IBUFF, OBUFF, IBP, OBP, ITRAN, OTRAN, MARGIN
COMMON /CONTRL/ CONTRL(64),IFILE,OFILE
COMMON /CONTRL/ CONTRL(64),IFILE,OFILE
INTEGER MAXPAC,WDSIZE,LINENB
COMMON /INIT/ MAXPAC,WDSIZE,LINENB
INTEGER SHL,SHR,SHLCNT,WORD,I
SHLCNT = 0
DO 100 J=1,I
  K = SHL(WORD,CONTRL(19)-6*J)
  IF(K .GE. 0) GO TO 80
  IF((K .GE. 0).OR.(SHLCNT.GE.6)) GO TO 60
  SHLCNT = SHLCNT + 1
  K = SHL(WORD,CONTRL(19)+SHLCNT-6*J)
  GO TO 50
  K = SHR(K,CONTRL(19)+SHLCNT-6)
  DO 70 L=1,SHLCNT
    K = K + 2**(6-L)
  CONTINUE
  SHLCNT = 0
  GO TO 90
  K = SHR(K,CONTRL(19)-6)
  OBUFF(OBP-J+1) = K
CONTINUE
RETURN
END

```

50

60

70

80

90

100

SUBROUTINE CLOOP

```

*****
* CLOOP CONTROLS THE PROGRAM FLOW WHEN STACKING *****
* DECISIONS AND REDUCTION DECISIONS ARE MADE. *****
*****

INTEGER V(215), VLOC(47), VINDX(13), C1(57), C1TRI(1), PRTB(53),
1PRD1B(53), HDTB(53), PRTB(53), CONTC(53), LEFTC(1), LEFTI(30),
2CCNCT(1), TRIPI(30), PRIND(47), NSY, NT, VLEN, VIL, CIL, NC1TRI, PRTBL,
3PRD1BL, HDTBL, PRTBL, CONCL, LEFTCL, LEFTIL, CONTL, TRIPL, PRIL, PACK,
41CCNCT, I DENTV, NUMBV, STRV, CATV, NCV, EOFIL,
1LEFTC, LEFTI, CONCTI, TRIPI, PRIND, NSY, NT, VLEN, VIL, CIL, NC1TRI,
2PRD1BL, PRTBL, HDTBL, PRTBL, CONCL, LEFTCL, LEFTIL, CONTL, TRIPL, PRIL,
3PACK, I DENTV, NUMBV, STRV, CATV, NCV, EOFIL,
1INTEGR, TOKEN, SP, PSTACK(75), MSTACK, PRMASK(5), MP, MPPI
LOGICAL FAILSF, COMPIL
1CCNCT/STACKS/SP, PSTACK, MSTACK, PRMASK, MP, MPPI, FAILSF, COMPIL
1INTEGR, ACCLEN, ACCUM(99), NUMB, IDENT, EOFLAG, SPECL, STR, STYPE, TYPE,
1CCNCT, VAR(75), VARTOP, VARC(256), MVAR
1CCNCT, VAR, VARTOP, VARC, MVAR
1INTEGR, SFL, SHR, RIGHT
LOGICAL STACK
COMPIL = .TRUE.
10 IF(.NOT. COMPIL) GO TO 999
IF(.NOT. STACK(0)) GO TO 30

STACK MAY HAVE SET COMPIL TO FALSE

IF(.NOT. COMPIL) GO TO 999
SP = SP + 1
IF(SP .LT. MSTACK) GO TO 20
CALL ERROR(2)
GO TO 999
20 PSTACK(SP) = TOKEN

INSERT ACCUM INTO VARC

IF((TOKEN.NE.STRV).AND.(TOKEN.NE.NUMBV).AND.(TOKEN.NE.IDENTV))
1GO TO 45
VAR(SPI) = VARTOP
IF(ACCLEN .EQ. 0) GO TO 999
DC 40 I=1, ACCLEN
VARC(I, VARTOP) = ACCUM(I)
VARTOP = VARTOP + 1
IF(VARTOP .LE. MVAR) GO TO 40

```



```

CALL ERROR(3)
VARTOP = 1
CONTINUE
CONTINUE
CALL SCAN
GC TO 10
CALL REDUCE
GC TO 10
RETURN
END
40
45
30
999

```



```

C 100 LOGICAL FUNCTION STACK(Q)
C      INTEGER GETCL,SHL,SHR
C      INTEGER SP,PSTACK(75),MSTACK,PRMASK(5),MP,MFPI
C      LOGICAL FAILSF,COMPIL
C      COMMON/STACKS/SP,PSTACK,MSTACK,PRMASK,MP,MFPI,FAILSF,COMPIL
C      COMMON/V(215),VLUC(47),VINDEX(13),CI(57),CI TRI(1),PRTB(53),
1  PRTC(53),CONTC(53),LEFTI(30),LEFTI(30),PRTBL,
2  CCNTI(1),TRIPI(30),PRIND(47),NSY,NT,VLEN,VIL,CIW,CIL,NCIPL,PACK,
3  PRDTBL,HDITBL,PRLENL,CONCL,LEFTCL,LEFTIL,CONTL,TRIPL,PRIL,PACK,
4  TCKEN,I/ SYNTAX/V,VLEN,PROCV,SEMIV,EOFIL
C      COMMON/IDEN/IDENI,NUMBV,STRV,PROCV,SEMIV,EOFIL
1  LEFTC,LEFTI,CONTI,TRIPI,PRIND,NSY,NT,VLEN,VIL,CIW,CIL,NCI TRI,
2  PRTBL,PRDTBL,HDITBL,PRLENL,CONCL,LEFTCL,LEFTIL,CONTL,TRIPL,PRIL,
3  PACK,GETCL(PSTACK(SP),TOKEN)+1
C      GO TO (100,200,300,400),I
50  GO TO (100,200,300,400),I
C      ILLEGAL SYMBOL PAIR
C      CALL ERKOR(1)
C      CALL PRSYM(0,PSTACK(SP))
C      CALL PAD(1,1,1)
C      CALL PRSYM(1,TOKEN)
C      CALL WRITEL(2)
C      COMPIL = .FALSE.
C      RETURN
C
C      RETURN TRUE
C
C 200  STACK = .TRUE.
C      GO TO 999
C
C      RETURN FALSE
C
C 300  STACK = .FALSE.
C      GO TO 999
C      J = SHL(PSTACK(SP-1),16)+SHL(PSTACK(SP),8)+TOKEN
C      IU = NCI TRI+2
C      IL = 1
C      K = SHR(IU+IL,1)
C      JP = CI TRI(K)
C      IF(J .LT. JP) IU=K
C      IF(J .GE. JP) IL=K
C      IF((IU-IL) .GT. 1) GO TO 450
C
C      CHECK FOR MATCH
C
C      STACK = J .EQ. CI TRI(IL)
C      RETURN
C 999  END

```



```

INTEGER FUNCTION GETC1(I,J)
INTEGER SHL,SHR,RIGHT
INTEGER V(215)
1PRDTC(53),HDTB(47),VINDX(13),C1(57),C1TRI(1),PRTB(53),
2CCNTT(1),TRIPI(30),PRLEN(53),CNTC(53),LEFTC(1),LEFTI(30),
3PRDTC(1),HDTB(1),PRLEN(1),CONCL(47),NSY,NT,VLEN,C1W,C1L,NC1TRI,PRIBL,
4TCKEN,IDENTV,NUMBV,STRV,CATV,NCAIV,EOFIL,CONCL,TRIPL,PACK,
CUMMCN /SYNTAX/V,VLOC,CI,C1TRI,PRTB,PROTB,HDTB,PRLEN,CONT,
1LEFTC,LEFTI,CONCL,TRIPL,PRIND,NSY,NT,VLEN,VIL,C1W,C1L,NC1TRI,
2PRDTC,PROTB,HDTB,PRLEN,CONCL,LEFTC,LEFTI,CONCL,TRIPL,PRIL,
3PACK,TCKEN,IDENTV,NUMBV,STRV,CATV,NCAIV,EOFIL,
4TCKEN,IDENTV,NUMBV,STRV,CATV,NCAIV,EOFIL,
INTEGER SP,PSTACK(75),MSTACK,PRMASK(5),MP,MPPI
LOGICAL FAILSF,COMPI
CUMMCN/STACKS/SP,PSTACK,MSTACK,PRMASK,MP,MPPI,FAILSF,COMPI
K = (NT+1)*I+J
L = K/15+1
L = C1(L)
M = SHL(14-MOD(K,15),1)
GETC1 = RIGHT(SHR(L,M),2)
RETURN
END

```



```

LOGICAL FUNCTION PROK (PRD)
INTEGER SHL,SHR,RIGHT,GETC1
INTEGER SP,PSTACK(75),MSTACK,PRMASK(5),MP,MPPI
LOGICAL FAILSF,COMPI
COMMON/STACKS/SP,PSTACK,MSTACK,PRMASK,MP,MPPI,FAILSF,COMPI
INTEGER V(215),VLOC(47),VINDX(13),CI(57),CITRI(1),PRTB(53),
1PRTB(53),HUTB(53),PRTB(53),PRLEN(53),CONTC(53),LEFTI(30),
2CCNTI(1),TRIPI(30),PRIND(47),NSY,NT,VLEN,VIL,CIL,NCITRI,PRTBL,
3PRTBTL,HUTBL,PRLENL,CONCL,LEFTCL,LEFTIL,CONTL,TRIPL,PRIL,PACK,
4TCUKEN,IDENTV,NUMBV,STRV,CATV,NCATV,EFILE
COMMON /SYNTAX/V,VLOC,VINDX,CI,CITRI,PRLEN,CONTC,
1LEFTC,LEFTI,CONTL,TRIPI,PRIND,NSY,NT,VLEN,VIL,CIL,NCITRI,
2PRTBTL,PRTBTL,HUTBL,PRLENL,CONCL,LEFTCL,LEFTIL,CONTL,TRIPL,PRIL,
3PACK,TUKEN,IDENTV,NUMBV,STRV,CATV,NCATV,EFILE
INTEGER PRD

```

```

CHECK FOR EQUAL OR EMBEDDED RIGHT PARTS

```

```

I = CONTC (PRD) + 1
GO TO (100,200,300,400),I

```

```

NC CHECK REQUIRED

```

```

100 PRCK = .TRUE.
RETURN

```

```

RIGHT CONTEXT CHECK

```

```

200 PROK = GETC1(HUTB(PRD),TOKEN) .NE. 0
RETURN

```

```

LEFT CONTEXT CHECK

```

```

300 K = HUTB(PRD) - NT
L = PRLEN(PRD)
I = PSTACK(SP-L)
LP = LEFTI(K) + 1
IF (L .GT. LP) GO TO 350
DO 350 J=L,LP
IF (LEFTC(J) .NE. I) GO TO 350
PROK = .TRUE.
RETURN

```

```

350 CCNTINUE
PROK = .FALSE.
RETURN

```

```

CHECK TRIPLES

```



```

C 400 K = HDTB(PRD) -- NT
    L = PRLN(PRD)
    I = SHL(PSTACK(SP-L),8) + TOKEN
    L = TRIPI(K) + 1
    LP = TRIPI(K+1)
    IF(L .LT. LP) GO TO 450
    DC 450 J=L+LP
    IF(CONT(J).NE. I) GO TO 450
    PRCK = .TRUE.
    RETURN
450 CCNTINUE
    PRCK = .FALSE.
    RETURN
    END

```



```

SUBROUTINE REDUCE
  INTEGER PRD,SHL,SHR,RIGHT,J,M
  INTEGER SP,PSTACK(75),MSTACK,PRMASK(5),MP,MFPI
  LOGICAL FAILSF,COMPIL
  COMMON/STACKS/SP,PSTACK,MSTACK,PRMASK,MP,MFPI,FAILSF,COMPIL
  INTEGER V(215),VLOC(47),VINDX(13),CI(57),CITRI(1),PRTB(53),
  1PRTB(53),HDTB(53),PRLEN(53),CONTC(53),LEFTC(1),LEFTI(30),
  2CONTT(1),TRIP(30),PRIND(47),NSY,NT,VLEN,VIL,CIW,NCITRI,PRTBL,
  3PRTBL,HDTB,PRLENL,STRV,PROCV,SEMI,EOFIL
  4TOKEN,I,IDENTV,NUMBV,CCNCL,LEFTC,LEFTI,CCNTL,TRIPL,PRIL,PACK,
  COMMON /SYNTAX/V,VLOC,VINDX,C1,CITRI,PRTB,PRDIB,HDTB,PRLEN,CONTC,
  1LEFTC,LEFTI,CONTT,TRIP,PRIND,NSY,LEFTC,LEFTI,CCNTL,TRIPL,PRIL,
  2PRTBL,PRTBL,HDTB,PRLENL,STRV,PROCV,SEMI,EOFIL
  3PACK,TOKEN,IDENTV,NUMBV,CCNCL,LEFTC,LEFTI,CCNTL,TRIPL,PRIL,
  LOGICAL JL,ML,PROK
  EQUIVALENCE (J,JL),(M,ML)

```

```

C
C
C      PACK THE TOP OF THE STACK

```

```

      K = SP - 4
      L = SP - 1
      J = 0
      DO 10 I=K,L
        I = SHL(J,8) + PSTACK(I)
        10 CONTINUE
        K = PRIND(PSTACK(SP)) + 1
        L = PRIND(PSTACK(SP)+1)
        DO 20 PRD=K,L
          M = PRLEN(PRD)
          ML = PRMASK(M)
          IF (M .AND. JL)
            IF (M .NE. PRTB(PRD)) GO TO 20
            IF (.NOT. PROK(PRD)) GO TO 20
          MP = SP - PRLEN(PRD) + 1
          MPPI = MP + 1
          J = HDTB(PRD)
          CALL SYNTH(PRTB(PRD),J)
          SP = MP
          PSTACK(SP) = J
          RETURN
        20 CONTINUE

```

```

C
C
C      NC APPLICABLE PRODUCTION
      FAILSF = .FALSE.
      CALL ERROR(4)
      COMPIL = .FALSE.
      RETURN
      END

```



```

210 ACCLEN = ACCLEN + 1
   ACCUM(ACCLEN) = I
   I = GNC(O)
   IF(I.GE. 2).AND.(I.LE. 11) GO TO 210
   IBP = IBP - 1
   TCKEN = NUMBV
   GC TO 5600
CC
CC
CC
IDENTIFIER
300 TYPE = IDENT
310 ACCLEN = ACCLEN + 1
   ACCUM(ACCLEN) = I
   I = GNC(O)
   IF(I.GE. 2).AND.(I.LE. 38) GO TO 310
   IBP = IBP - 1
   IF(ACCLEN.LE. 6) GO TO 315
   CALL ERROR(7)
   CALL FORM(0,ACCUM,1,ACCLEN,ACCLEN,.FALSE.)
   CALL WRITEL(1)
   CUMFIL = .FALSE.
   RETURN
315 STYPE = 0
CC
CC
CC
GO LOOK FOR SPECIAL TOKENS
GC TO 5000
CC
CC
CC
SPECIAL CHARACTER FOUND
400 IF(I.EQ. 46) GO TO 500
   IF(I.EQ. 41) GO TO 450
   TYPE = SPECL
   ACCLEN = 1
   ACCUM(1) = I
   GC TO 5000
CC
CC
CC
BEGINNING OF CONCATINATION FOUND
450 TYPE = CONCAT
   I = GNC(O)
   IF(I.EQ. 41) GO TO 470
   IF(I.NE. 45) GO TO 460
   TCKEN = NCATV
   I = GNC(O)
   IF(I.EQ. 41) GO TO 5600
   CALL ERROR(5)
   CUMFIL = .FALSE.
460

```



```

470 RETURN = CATV
    TOKEN = 5600
C
500 STRING QUOTE FOUND
    TYPE = STR
    ACCUM(1) = 1
510 I = GNC(0)
    IF(I.EQ. 46) GO TO 530
520 ACCLEN = ACCLEN + 1
    IF(ACCLEN.LE. 99) GO TO 525
    CALL ERROR(6)
    RETURN
525 ACCUM(ACCLEN) = I
    GO TO 510
C
    STRING QUOTE FOUND, CHECK FOR END
C
530 I = GNC(0)
    IF(I.EQ. 46) GO TO 520
    IBP = IBP - 1
    STYPE = 0
    TOKEN = STRV
    GO TO 5600
C
    CHECK FOR SPECIAL TOKENS
C
5000 TOKEN = 0
    J = VINDX(ACCLEN) + 1
    K = VINDX(ACCLEN+1)
    DC 5100 I=J,K
        L = VLOC(I)
        LP = L + V(L)
        L = L + 1
        N = 1
        DO 5200 M=L,LP
            IF(ACCUM(N) .NE. V(M)) GO TO 5100
            N = N + 1
        CONTINUE
        TOKEN = I - 1
        GO TO 5600
5200 CONTINUE
5100 CONTINUE
    GO TO 5500
5300 GO TO 5600
5500 IF(TYPE .NE. IDENT) GO TO 5600
5600 TOKEN = IDENTV
    RETURN
    END

```



```

SLBROUT INE SYNTH (PROD, SYM)
INTEGER /IBUFF(80), OBUFF(120), IBP, OBP, ITRAN(256), QTRAN(64), MARGIN
INTEGER SP, PSTACK(75), MSTACK, PRMASK(5), MP, MPPI
LOGICAL FAILSF, COMPILE
COMMON /V(215), VLCC(47), VINDEX(13), C1(37), C1TRI(1), PRIB(53),
1 PRDTB(53), HDTB(53), PKLEN(53), CONTC(53), LEFTC(1), LEFTI(30),
2 CONTT(1), TRIPI(30), PRIND(47), NSY, NT, VLEN, VIL, C1L, NC1TRI, PRIBL,
3 PRDTBL, HDTBL, PRLENL, CONCL, LEFTCL, LEFTIL, CONTL, TRIPL, PRIL,
4 TCKEN, IDENTV, NUMBV, STRV, CATV, NCATV, EOFIL
1 LEFTC, LEFTI, CONTT, TRIPI, PRIND, NSY, NT, VLEN, VIL, C1L, NC1TRI,
2 PRDTBL, HDTBL, PRLENL, CONCL, LEFTCL, LEFTIL, CONTL, TRIPL, PRIL,
3 PACK, TCKEN, IDENTV, NUMBV, STRV, CATV, NCATV, EOFIL
1 CCNCAT, VAR(75), VARTOP, VARC(256), MVAR
1 COMMON /SCANN/ACCLEN, ACCUM, NUMB, IDENT, ECFLAG, SPECL, STF, STYPE, TYPE,
1 CCNCAT, VAR, VARTOP, VARC, MVAR
1 COMMON /INIT/ MAXPAC, WDSIZE, LINENB
1 INTEGER /VERLEN, START, FINISH, LENGTH, PAKCNT, VERSE(500), CC
1 COMMON /FORMS/ VERLEN, START, FINISH, LENGTH, PAKCNT, VERSE, CC
1 INTEGER /RADIX(8), VARB(8), FW(8)
1 COMMON /CNOUT/ RADIX, VARB, FW
1 INTEGER /EQUIV(10), EOPTR, NUMCTR, NUMLN, MAXLN, CBPCK
1 COMMON /EQUIVA/ EQUIV, EOPTR, NUMCTR, NUMLN, MAXLN, CBPCK
1 COMMON /TEXTS/ TEXT
1 COMMON /AFLAG, TOGSET
1 LOGICAL /VCCPY(37), COMLEN(37), COMVEC(500), CPTR, SOURCE(6), PCNT,
1 COMMON /COPYS/ VCCPY, COMLEN, COMVEC, CPTR, SOURCE, PCNT, ERFLAG, SORCE,
1 INTEGER PROD, SYM, SHL, SHR, RIGHT, CCNTIN, BASE
1 GC TO (100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300,
1 1400, 1500, 1600, 1700, 1800, 1900, 2000, 2100, 2200, 2300, 2400,
1 2500, 2600, 2700, 2800, 2900, 3000, 999, 999, 999, 999, 999,
1 999, 999, 999, 999, 999, 999, 999, 999, 999, 999, 999, 999, 999,
1 5200), PROD
1 <PROGRAM> ::= <STATEMENT LIST>
1 COMPILE = .FALSE.
1 IF (AFLAG) GC TO 999
1 OUTPUT(1) CALL WRITEL(0, 1)
1 CALL FORM(0, TEXT, 12, 15, 20, .TRUE.)
1 CALL WRITEL(0)

```

C 100
C
C


```

C 200      CC = 0
          GC TO 999
          <STATEMENT LIST> ::= <STATEMENT>
          CCMPIL = FALSE
          IF (AFLAG) GO TO 999
          C  OUTPUT(, CALL WRITEL(0,))
          CALL FORM(0,TEXT,12,15,20,.TRUE.)
          CALL WRITEL(0)
          CC = 0
          GC TO 999
          <STATEMENT LIST> ::= <STATEMENT LIST> <STATEMENT>
          C 300      CCNTINUE
          GC TO 999
          <STATEMENT> ::= <OUTPUT_STATEMENT>
          C 400      CCNTINUE
          GC TO 999
          <STATEMENT> ::= <APPEND_STATEMENT>
          C 500      CCNTINUE
          GC TO 999
          <STATEMENT> ::= <COPY_STATEMENT>
          C 600      CCNTINUE
          GC TO 999
          <STATEMENT> ::= <CHANGE_STATEMENT>
          C 700      CCNTINUE
          GC TO 999
          <STATEMENT> ::= <INSERT_STATEMENT>
          C 800      CCNTINUE
          GC TO 999
          <STATEMENT> ::= <DELETE_STATEMENT>
          C 900      CCNTINUE
          GC TO 999
          <STATEMENT> ::= <COPY> <NUMBER>
          C 1000     J = VARTOP
          K = VARTOP - 1
          II = 0
          VARTOP = VAR(SP)
          N = K - J
          DC 1010 I = J,K
          II = (VARC(I)-2)*10**N+II
          N = N - I
          C 1010     CCNTINUE
          START = VCOPY(II)
          LENGTH = COMLEN(II)
          FINISH = COMFIN(II)
          CALL FORM(0,COMVEC,START,FINISH,LENGTH,.TRUE.)
          CALL WRITEL(0)
          AFLAG = TRUE.
          GC TO 999

```



```

1222 CALL FORM(0,TEXT,43,44,6,.TRUE.)
C      J=J+ EQUIV(I)
      OUTPUT(//,(VERSE(//(-10) J:10//'),VERSE'/'/(-4) JJ:10//') I:10//)
      CALL FORM(1,TEXT,45,46,7,.TRUE.)
      CALL CONOUT(2,-10,J,10)
      CALL FORM(1,TEXT,47,48,7,.TRUE.)
      CALL CALL CONOUT(2,-4,I,10)
C      OUTPUT(//,(1)///)
      CALL FORM(1,TEXT,49,49,4,.TRUE.)
      IF (I.EQ. K) GO TO 1220
C      OUTPUT(//,///)
      CALL FORM(1,TEXT,34,34,1,.TRUE.)
      CONTINUE
      CALL WRITEL(0)
      KK=0
1225 DC 1230 J=1,EQPTR
      IF (EQPTR.EQ. 1) GO TO 1235
      JJ=J-1
      OUTPUT( DATA VERSE'/'/(-4) JJ:10//')
C      CALL FORM(0,TEXT,50,53,16,.TRUE.)
      CALL CONOUT(2,-4,JJ,10)
      CALL FORM(1,TEXT,54,54,1,.TRUE.)
      CALL CALL WRITEL(0)
      GO TO 1236
C      OUTPUT( DATA VERSE//')
1235 CALL FORM(0,TEXT,55,58,17,.TRUE.)
      CALL CALL WRITEL(0)
C      OUTPUT( *//-'//)
1236 CALL FORM(0,TEXT,59,60,6,.TRUE.)
      IF (J.EQ. 1) GO TO 1237
      KK=KK+ EQUIV(J)
      K= EQUIV(J)
      DO 1240 I=1,K
      IVAL=VERSE(I+KK)
      OUTPUT(//(-20) IVAL:10//) ,10
      CALL CONOUT(2,-20,K) GO TO 1240
      IF (I.EQ. K) GO TO 1240
      OUTPUT(//,///)
C      CALL FORM(1,TEXT,34,34,1,.TRUE.)
1240 CONTINUE
      IF (J.EQ. EQPTR) GO TO 1230
      OUTPUT(//,///)
C      CALL FORM(1,TEXT,61,61,1,.TRUE.)
1230 CONTINUE
      J=500-NTGT
      OUTPUT(//,///)
C      CALL FORM(1,TEXT,34,34,1,.TRUE.)
C      CALL FORM(1,TEXT,34,34,1,.TRUE.)
      OUTPUT( *//(-4)J:10//,*0//)

```



```

CALL FCRM(0,VERSE,16,17,6,.TRUE.)
CALL CONOUT(1,-4,J,10)
CALL FCRM(1,VERSE,18,18,3,.TRUE.)
CALL WRITEL(0)
GC TO 999
C 1300 <OUTPUT_HEAD> ::= <OUTPUT> (
CONTINUE
GC TO 959
C 1400 <OUTPUT_HEAD> ::= <OUTPUT_HEAD> <STRING>
VERLEN = VERLEN + 1
AFLAG = .FALSE.
START = VERLEN
LENGTH = 0
PAKCNT = 1
J = VARTOP - 1
K = VARTOP = VAR(SP)
DC 20 I=J,K
IF(PAKCNT.LE.MAXPAC) GO TO 10
PAKCNT = VERLEN + 1
KV = SHL(VERSE(VERLEN),6)
VERSE(VERLEN) = KV + VARC(I)
PAKCNT = PAKCNT + 1
LENGTH = LENGTH + 1
CONTINUE = VERLEN
FINISH = VERLEN
C 20 OUTPUT(//, '-/(-4)CC:10//',VERSE,'-/(-4)START:10//')
CALL FCRM(0,TEXT,62,65,16,.TRUE.)
CALL CONOUT(1,1,CC,10)
CALL FCRM(1,TEXT,66,67,7,.TRUE.)
CALL CONOUT(2,-4,START,10)
CALL FCRM(1,TEXT,68,68,1,.TRUE.)
CALL CONOUT(2,-4,FINISH,10)
CALL FCRM(1,TEXT,69,69,1,.TRUE.)
CALL CONOUT(2,-4,LENGTH,10)
CALL FCRM(1,TEXT,70,71,8,.TRUE.)
CALL WRITEL(0)
BASE = 10
DC 30 I=START,FINISH
J = VERSE(I)
NDIGIT = 1
J = J / BASE
IF (J.EQ.0) GO TO 33
NDIGIT = NDIGIT + 1
GO TO 31
JJ = OBPK + NDIGIT + 1
C 31
C 33

```



```

IF (JJ.LE. 71) GO TO 37
JJJ = NUMLN + 1
IF (JJJ.LE. MAXLN) GO TO 35
NUMLN = 0
EQPTR = EQPTR + 1
EQUIV(EQPTR) = NUMCTR
NUMCTR = 0
OBPCK = 6
NUMLN = NUMLN + 1
OBPCK = OBPCK + NDIGIT + 1
NUMCTR = NUMCTR + 1
35 CCNT INUE
GC TO 999
C 1500 <OUTPUT_HEAD> ::= <OUTPUT_HEAD> <CONCAT>
CC=1
AFLAG = .TRUE.
GC TO 999
C 1600 <OUTPUT_HEAD> ::= <OUTPUT_HEAD> <NCONCAT>
CC=2
AFLAG = .TRUE.
GC TO 999
C 1700 <OUTPUT_HEAD> ::= <OUTPUT_HEAD> <CONOUT_CALL>
CCNT INUE
C 1700 <CONOUT_CALL> ::= <CONOUT_CALL>
C 1700 <CONOUT_CALL> ::= <CONOUT_CALL>
C 1700 CALL FORM(0,TEXT,72,75,18,.TRUE.)
C 1700 CALL CONOUT(2,-3,CC,10)
C 1700 CALL PAD(1,48,1)
C 1700 CALL FORM(1,FW,1,6,8,.FALSE.)
C 1700 CALL PAD(1,48,1)
C 1700 CALL FORM(1,VARB,1,6,8,.FALSE.)
C 1700 CALL PAD(1,43,1)
C 1700 CALL FORM(1,RADIX,1,6,8,.FALSE.)
C 1700 CALL PAD(1,43,1)
C 1700 CALL WRITEL(0)
C 1700 AFLAG = .FALSE.
GC TO 999
C 1800 <CONOUT_CALL> ::= <CONOUT_STATEMENT>
CCNT INUE
GC TO 999
C 1900 <CONOUT_CALL> ::= <CONOUT_CALL> <RADIX>
CCNT INUE
GC TO 999
C 2000 <CONOUT_STATEMENT> ::= <IDENTIFIER>
C 2000 GC TO 2200
C 2100 <CONOUT_STATEMENT> ::= <NUMBER>
C 2100 GC TO 2200

```



```

C 2200      <CONCAT STATEMENT> ::= <FIELD_WIDTH> <IDENTIFIER>
          J = VAR(SP)
          K = 1
          L = VARTOP - 1
          DC 13901 I=J,L
              VARB(K) = VARC(I)
              K = K + 1
13901      CCNTINUE
          IJ = VARTOP - J + 1
          DC 13902 I=IJ,8
              VARB(I) = I
13902      CCNTINUE
          VARTOP = VAR(SP)
          GC TO 999
          <CONCAT STATEMENT> ::= <FIELD_WIDTH> <NUMBER>
C 2300      GC TO 2200
C 2400      <FIELD_WIDTH> ::= <FW_HEAD> )
          CCNTINUE
          GC TO 999
          <FW_HEAD> ::= (
C 2500      CCNTINUE
          GC TO 999
          <FW_HEAD> ::= <FW_HEAD> <IDENTIFIER>
C 2600      J = VAR(SP)
          K = 1
          L = VARTOP - 1
          DC 14301 I=J,L
              FW(K) = VARC(I)
              K = K + 1
14301      CCNTINUE
          IJ = VARTOP - J + 1
          DC 14302 I=IJ,8
              FW(I) = I
14302      CCNTINUE
          VARTOP = VAR(SP)
          GC TO 999
          <FW_HEAD> ::= <FW_HEAD> <NUMBER>
C 2700      GC TO 2600
C 2800      <RADIX> ::= <RADIX_HEAD> <NUMBER>
          J = VAR(SP)
          K = 1
          L = VARTOP - 1
          DC 14501 I=J,L
              RADIX(K) = VARC(I)
              K = K + 1
14501      CCNTINUE
          IJ = VARTOP - J + 1
          DC 14502 I=IJ,8

```



```

14502 RADIX(I) = 1
      CCNTINUE
      VARTOP = VAR(SP)
      GC TO 999
      <RADIUS> ::= <RADIUS_HEAD> <IDENTIFIER>
2900   GO TO 2800
      C
3000   <RADIUS_HEAD> ::= :
      CCNTINUE
      GC TO 999
      <OUTPUT> ::= OUTPUT
      C
4900   CCNTINUE
      GO TO 999
      <APPEND> ::= APPEND
      C
5000   CCNTINUE
      GO TO 999
      <COPY> ::= COPY
      C
5100   CCNTINUE
      GC TO 999
      <SUBJECT> ::= VERSE
      C
5200   CCNTINUE
      999 RETURN
      END

```



```

SUBROUTINE MERG
INTEGER CONTRL(64), IFILE, OFILE
COMMON /CONTRL/ CONTRL, IFILE, OFILE
INTEGER IBUFF(80), OBUFF(120), IBP, OBP, ITRAN(256), OTRAN(64), MARGIN
COMMON /FILES/ IBUFF, OBUFF, IBP, OBP, ITRAN, CTRAN, MARGIN
INTEGER PABUFF(80), SOBUFF(80), SONUM(8), PANUM(8), PASNUM(8), MERGE
LOGICAL INSERT, FAULT
COMMON /MERGES/ PABUFF, SOBUFF, SONUM, PANUM, PASNUM, MERGE, INSERT,
1 FAULT
LOGICAL NINES, MERGIT
50 IF(PABUFF(9) .NE. 47) GO TO 200
CALL SEGNUM(2)
100 IFILE = CONTRL(20)
150 CALL RUTEXT(2)
CALL SEGNUM(3)
IF(MERGIT(SONUM, PASNUM)) GO TO 150
IFILE = CONTRL(1)
CALL RUTEXT(3)
CALL SEGNUM(1)
IF(NINES(1)) CONTRL(24) = 0
IF(MERGIT(PANUM, SONUM)) GO TO 50
DC 175 I = 1, 80
175 IBUFF(I) = SOBUFF(I)
CCONTINUE
RETURN
200 INSERT = .TRUE.
DC 210 I = 1, 72
210 IBUFF(I) = PABUFF(I+8)
CCONTINUE
DC 220 I = 73, 80
220 IBUFF(I) = PABUFF(I-72)
CCONTINUE
RETURN
END

```



```

SUBROUTINE SEQDK
  INTEGER I8UFF(30), QBUFF(120), IBP, OBP, ITRAN(256), OTRAN(64), MARGIN
  COMMON /FILES/ IBUFF, QBUFF, IBP, OBP, ITRAN, CTRAN, MARGIN
  INTEGER CONTRL(64), IFILE, OFILE
  COMMON /CONTRL/ CONTRL, IFILE, OFILE
  INTEGER SEQN
  LOGICAL SEQ
  COMMON /SEQNS/ SEQN, SEQ
  IF (SEQ) GO TO 10
  SEQ = .TRUE.
  SEQN = CONTRL(13)
  GO TO 11
10 SEQN = SEQN + CONTRL(15)
11 ISEQN = SEQN
  DO 131 I=1,8
    IBUFF(81-I) = MOD(ISEQN,10) + 2
    ISEQN = ISEQN/10
131 CONTINUE
  RETURN
  END

```



```

SUBROUTINE COPY
  INTEGER VCOPY(37), COMLEN(37), COMVEC(500), CPTR, SOURCE(6), PCNT,
1  LOGICAL ERFLAG, SORCE
  COMMON /COPYS/ VCOPY, COMLEN, COMVEC, CPTR, SOURCE, PCNT, ERFLAG, SORCE,
1  INTEGER MAXPAC, WDSIZE, LINENB
  COMMON /INIT/ MAXPAC, WDSIZE, LINENB
  COMMON /IBUFF(80), OBUFF(120), IBP, CBP, ITRAN(256), CTRAN(64), MARGIN
  COMMON /FILES/ IBUFF, OBUFF, IBP, CBP, ITRAN, CTRAN, MARGIN
  INTEGER /CNTRL(64), IFILE, OFILE
  COMMON /CNTRL/ CNTRL, IFILE, OFILE
  INTEGER PABUFF(80), SUBUFF(80), SCNUM(8), PANUM(8), PASNUM(8), MERGE
  LOGICAL INSERT, FAULT
  N = CNTRL(19)/6
  J = IBUFF(3) - 2
  IF (IBUFF(4) .EQ. 1) GO TO 30
  J = J*10 + (IBUFF(4) - 2)
30 DC 10 I = 1, 5
  IBUFF(1) = 1
10 CONTINUE
  CPTR = CPTR + 1
  VCOPY(J) = CPTR
  L = 0
  PCNT = 1
40 DC 50 I = 1, 80
  IF (PCNT .LE. N) GO TO 45
  PCNT = 1
  CPTR = CPTR + 1
  K = SHL(COMVEC(CPTR), 6)
  COMVEC(CPTR) = K + IBUFF(1)
  PCNT = PCNT + 1
  L = L + 1
50 CONTINUE
  COMFIN(J) = CPTR
  IBP = 81
  I = GNC(0)
  IF (CNTRL(24) .EQ. 0) GO TO 60
  IF (SUBUFF(1) .NE. CNTRL(31)) GO TO 40
  COMLEN(J) = L
  DC 55 I = 1, 80
  IBUFF(1) = SUBUFF(1)
55 CONTINUE
  GO TO 999
60 IF (IBUFF(1) .NE. CNTRL(31)) GO TO 40
  COMLEN(J) = L
999 RETURN
END

```



```

SUBROUTINE SETTOS
  INTEGER IBUFF(80), OBUFF(120), IBP, OBP, ITRAN(256), OTRAN(64), MARGIN
  COMMON /FILES/ IBUFF, OBUFF, IBP, OBP, ITRAN, OTRAN, MARGIN
  INTEGER CONTRL(64), IFILE, OFILE
  COMMON /CONTRL/ CONTRL, IFILE, OFILE
  INTEGER VCOPY(37), COMLEN(37), COMVEC(500), CPTR, SOURCE(6), PCNT,
1  LOGICAL COMFIN(37)
  LOGICAL ERFLAG, SOURCE
  COMMON /COPYS/ VCOPY, COMLEN, COMVEC, CPTR, SOURCE, PCNT, ERFLAG, SOURCE,
1  LOGICAL AFLAG, TOGSET
  COMMON /FLAGS/ AFLAG, TOGSET
  INTEGER PATCH(5)
  DATA PATCH/27,12,31,14,19/
  JJ = 3
  IF (IBUFF(JJ) .NE. 1) GO TO 10
  IF (IBUFF(JJ) .NE. 1) GO TO 7
  JJ = JJ + 1
  KK = JJ
  GO TO 5
7  DC 9 I = 1, 6
  IF (SOURCE(I) .NE. IBUFF(JJ)) GO TO 15
  JJ = JJ + 1
9  CONTINUE
  SCRCN = .TRUE.
  RETURN
15  DC 16 I = 1, 5
  IF (PATCH(I) .NE. IBUFF(KK)) GO TO 999
  KK = KK + 1
16  CONTINUE
  TCGSET = .TRUE.
  RETURN
10  J = IBUFF(3)
  DC 20 I = 4, 80
  II = I
  IF (IBUFF(I) .EQ. 39) GO TO 30
20  CONTINUE
  GC TO 999
30  K = 0
  I = I + 1
  DC 40 I = I, 80
  L = IBUFF(I)
  IF (L .LE. 11) GO TO 40
  IF (L .GT. 11) GO TO 50
  K = K*10 + (L-2)
40  CONTINUE
50  CCNTRL(J) = K
  RETURN

```



```
999 CALL ERROR(8)  
      ERFLAG = .TRUE.  
      RETURN  
      END
```



```

LCGICAL FUNCTION NINES(K)
INTEGER PABUFF(80), SCBUFF(80), SCNUM(8), PANUM(8), PASNUM(8), MERGE
LCGICAL INSERT, FAULT
COMMON /MERGES/ PABUFF, SOBUFF, SONUM, PANUM, PASNUM, MERGE, INSERT,
1 FAULT
GC TO (10,20),K
10 DC 15 I=1,8
15 IF(PANUM(I) .NE. 11) GO TO 990
15 CCNTINUE
NINES = .TRUE.
GC TO 999
20 DO 25 I=1,8
25 IF(SONUM(I) .NE. 11) GO TO 990
25 CCNTINUE
NINES = .TRUE.
GC TO 999
990 NINES = .FALSE.
999 RETURN
END

```

```

FUNCTION IMIN(I,J)
IF(I .LT. J) GO TO 10
IMIN = J
GC TO 999
10 IMIN = I
RETURN
999 END

```

```

FUNCTION ICON(I)
INTEGER IBUFF(80), OBUFF(120), IBP, OBP, ITRAN(256), OTRAN(64), MARGIN
COMMON /FILES/ IBUFF, OBUFF, IBP, OBP, ITRAN, CTRAN, MARGIN
DC 100 K=1,52
J = K
IF( I .EQ. CTRAN(K) ) GO TO 200
100 CCNTINUE
J = 1
100 ICON = J
200 RETURN
END

```



```
INTEGER FUNCTION SHL(I,J)
SHL = I*(2**J)
RETURN
END
```

```
INTEGER FUNCTION SHR(I,J)
SHR = I/(2**J)
RETURN
END
```

```
INTEGER FUNCTION RIGHT(I,J)
RIGHT = MOD(I,2**J)
RETURN
END
```



```

SUBROUTINE ERROR(I)
  INTEGER MAXPAC,WDSIZE,LINENB
  COMMON /INIT/ MAXPAC,WDSIZE,LINENB
  INTEGER /TEXT(108)
  COMMON /TEXTS/ TEXT
  LINE = LINENB - 1
  OUTPUT(,***** ERROR('/(2)I:10//') NEAR LINE '/-/(5)LINE :10)
  CALL FORM(0,TEXT ,80,82,12,,.TRUE.)
  CALL CONOUT(1,2 ,1 ,10 ,)
  CALL FORM(1,TEXT ,83,85,12,,.TRUE.)
  CALL CONOUT(2,-5 ,LINE ,10 ,)
  CALL WRITEL(0)
  GO TO(100,200,300,400,500,600,700,800),I
100 CONTINUE ***** ILLEGAL SYMBOL PAIR. SYMBOLS ARE LISTED BELOW')
  OUTPUT(,FORM(0,TEXT ,86,96,51,.TRUE.)
  CALL WRITEL(0)
  GO TO 999
200 CONTINUE ***** PARSE STACK OVERFLOW')
  OUTPUT(,FORM(0,TEXT ,97,102,26,.TRUE.)
  CALL WRITEL(0)
  GO TO 999
300 CONTINUE ***** TABLE OVERFLOW')
  OUTPUT(,FORM(0,TEXT ,103,106,20,.TRUE.)
  CALL WRITEL(0)
  GO TO 999
400 CONTINUE ***** IMPROPERLY FORMED STATEMENT')
  OUTPUT(,FORM(0,TEXT ,107,113,33,.TRUE.)
  CALL WRITEL(0)
  GO TO 999
500 CONTINUE ***** INCORRECT FORMAT FOR CONCATENATION USED')
  OUTPUT(,FORM(0,TEXT ,114,122,45,.TRUE.)
  CALL WRITEL(0)
  GO TO 999
600 CONTINUE ***** STRING LENGTH EXCEEDS LIMITS')
  OUTPUT(,FORM(0,TEXT ,123,129,34,.TRUE.)
  CALL WRITEL(0)
  GO TO 999
700 CONTINUE ***** VARIABLE BELOW EXCEEDS FORTRAN LENGTH LIMITS')
  OUTPUT(,FORM(0,TEXT ,130,139,50,.TRUE.)
  CALL WRITEL(0)
  GO TO 999

```



```
C      800  CCNTINUE  
      C      COUTPUT(,***** IMPROPERLY FORMED $$CONTROL STATEMENT.)  
      C      CALL FORM(0,TEXT ,140,148,43,.TRUE.)  
      C      CALL WRITEL(0)  
      C      RETURN  
      999  END
```



```

DATA TEXT0/
*17043521,23974927,343779184,254953390,342462531,672665803,
*152571969,17043521,65,17043521,20497879,25810207,
*274636971,17043521,47,192681,17043521,65,17043521,20555288,
*442767967,277740097,557963152,17043521,22124496,4785985,17043521,
*7,557963152,42,43,48,17043521,221088276,556889113,239082145,
*276161578,61802576,494469290,965360,17043521,47,1713361246,1066,
*724139421,1936,11025131,17043521,20760524,254953390,16,41,
*17043521,20760524,254953390,1065,17043521,47,41,17043521,20457879,
*21342040,42,814024542,1072,48,48,815920992,68139,17043521/
DATA TEXT1/
*26497879,20555354,133098,17043521,20497879,20555354,133098,
*801045487,21092186,1898,172185868,486896521,1025,801045487,
*22115792,305229918,6103226167,23905565,871475992,225015681,
*206894391,343536655,20252122,34,801045487,23906142,268822476,
*240653985,276108762,34,801045487,24953687,268806224,491091618,
*801045487,22120157,443352919,604313245,406909022,523367448,67167,
*801045487,22123418,494207903,21342040,209458266,486860441,
*238154777,209536665,25285743,801045487,24770388,424154576,
*424277185,277668880,259523143,6375390,801045487,25478996/
DATA TEXT2/
*204829697,222394018,21115792,272490577,443928636,419787801,
*310194263,341919710,801045487,22120157,443352919,604313245,
*406909020,841312351,493449310,523367448,67167/
INTEGER VCOPY(37),COMLEN(37),COMVEC(500),CPTR,SOURCE(6),PCNT,
1 LCGICAL COMFIN(37)
1 LCGICAL ERFLAG,SOURCE
1 COMMON /COPYS/ VCOPY,COMLEN,COMVEC,CPTR,SOURCE,PCNT,ERFLAG,SORCE,
1 DATA SOURCE/30,26,32,29,14,16/,VCOPY/37*0/,COMLEN/37*0/,
1 INTEGER COMVEC/500*0/,SORCE/,FALSE/,ERFLAG/,FALSE/,CPTR/0/
1 COMMON /RADIX/ RADIX,VARB,FW(8)
1 COMMON /CNCUT/ RADIX,VARB,FW
DATA RADIX/8*0/,RADIX,VARB,FW
1 INTEGER V(212),VLOC(47),VINDEX(13),C1(57),C1TRI(1),PRIB(53),
1PRIB(53),HDIR(53),PRLEN(53),CONTC(53),LEFTC(1),LEFTI(30),
2CONTT(1),TRIPI(30),PRIND(47),NSY,NT,VLEN,VIL,C1W,C1L,NC1TRI,PRIBL,
3PRCKEN,IDENTV,NUMBV,STRV,CATV,NCATV,LEFTCL,LEFTIL,CONTL,TRIPL,PACK,
4COMMON /SYNTAX/ V,VLOC,CC,C1TRI,PRIB,PRDTB,HDTB,PRLEN,CONTC,
1LEFTC,LEFTI,CONTT,TRIPI,PRIND,NSY,NT,VLEN,VIL,C1W,C1L,NC1TRI,
2PRIBL,PRCKEN,IDENTV,NUMBV,STRV,CATV,NCATV,LEFTCL,LEFTIL,CONTL,TRIPL,PRIL,
3INTEGER FOR VO(160),V1(55)
1EQUIVALENCE (V(1),VO(1)),(V(161),V1(1))
DATA VO/18,49,16,29,29,26,27,36,5,33,16,29,30,16,6,14,
11,42,1,51,2,31,26,3,16,26,17,4,14,26,27,36,5,33,16,29,30,16,6,14,
215,12,25,18,16,0,15,16,23,10,31,16,6,20,25,30,16,29,31,6,26,32,31,

```


LIST OF REFERENCES

1. Pool, P. C. and Waite, W. M., "Machine Independent Software," Procedures of the ACM Second Symposium on Operating Systems Principles, p. 19-24, October 1969.
2. Gries, D., Compiler Construction for Digital Computers, p. 64-83, Wiley, 1971.
3. Orgass, R. J. and Waite, W. M., "A Base for a Mobile Programming System," Communications of the ACM, Volume 12, Number 9, p. 507-510, September 1969.
4. McKeeman, W. M., Horning, J. J., and Wortman, D. B., a compiler generator, p. 7, Prentice-Hall, 1970.
5. National Technical Information Service, U. S. Department Of Commerce, Programming For Transferability, by J. E. Fleiss, et al, p. 1-44, 72-87, September 1972.
6. Pool, P. C. and Waite, W. M., "Portability and Adaptability," Advanced Course on Software Engineering, p. 183-277, 1973.
7. Halpern, M. I., "Toward a General Processor for Programming Languages," Communications of the ACM, Volume 11, Number 1, p. 15-25, January 1968.
8. Waite, W. M., "The Mobile Programming System: STAGE2," Communications of the ACM, Volume 13, Number 7, p. 415-421, July 1970.
9. Waite, W. M., "A Language Independent Macro Processor," Communications of the ACM, Volume 10, Number 7, p. 433-440, July 1967.
10. Denning, P. J., "Is It Not Time To Define "Structured Programming"?", Operating Systems Review, Volume 8, Number 1, p. 6-7, January 1974.
11. A Guide To PL/M Programming, Revision 1, Intel Corporation, September 1973.
12. Hamlet, R. G., Portability and Adaptability, computing reviews, p. 62-63, February 1974.

BIBLIOGRAPHY

1. Brown, P. J., "The ML/I Macro Processor," Communications of the ACM, Volume 10, Number 10, p. 618-623, October 1967.
2. Cook, A. J., "A User's Guide to MORTAN," SLAC Computation Group, Stanford, California, CGTM NO. 150, July, 1973.
3. Kildall, G. A., PL/M Compiler Pass-1 Version 2.0, Copyright Intel Corporation, 1973.
4. McIlroy, M. D., "Macro Instruction Extensions of Compiler Languages," Communications of the ACM, volume 3, Number 4, p. 214-220, April 1960.
5. Parnas, D. L., "A Technique for Software Module Specification with Examples," Communication of the ACM, Volume 15, Number 5, p. 330-336, May 1972.
6. Sibley, R. A., "The SLANG System," Communications of the ACM, Volume 4, Number 1, p. 75-84, January 1961.
7. Steel, T. B., "UNCOL," Datamation, Number 1, p. 18-20, January/February, 1960.

INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2. Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3. Chairman, Code 72 Computer Science Group Naval Postgraduate School Monterey, California 93940	1
4. LT Edward C. Coulter, USN 841 E Bantam Road Tucson, Arizona	1
5. LT Daniel J. Parker, USN 5722 Central Avenue Bonita, California	1
6. Prof. Gary A. Kildall, Code 72Kd Computer Science Group Naval Postgraduate School Monterey, California 93940	1

15 MAY 75
15 OCT 75
27 JAN 76

22956
23417
23499

Thesis

152564

C75665 Coulter
c.1

Machine independence:
the problem of port-
ability.

15 MAY 75
15 OCT 75
27 JAN 76

22956
23417
23499

JAN 5 '76

The
C75
c.1

Thesis

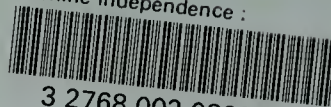
152564

C75665 Coulter
c.1

Machine independence:
the problem of port-
ability.

thesC75665

Machine independence :



3 2768 002 08983 1

DUDLEY KNOX LIBRARY